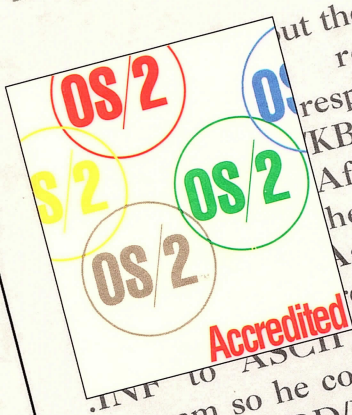
# OS/2® EXTRA!

# KBD, MOU, & VIO SPECIAL FUNCTIONS REVEALED




## Edited by Len Dorfman and Marc J. Neuberger

this book came into being because Len hates using
He likes to read while sitting at his
likes to read sitting in his living
He does not like to read

ut the
rel
respo
KBI
Afte
he
Asp
tc

.INF to ASCII
program so he cou
the VIO/KBD/M
With the help of
written by C
describing the
Marc took up
and a few days
ious h

19.95

# OS/2® Extra!

## KBD, MOU, & VIO special functions revealed

# OS/2® Extra!

## KBD, MOU, & VIO special functions revealed

*Edited by*
Len Dorfman
Marc J. Neuberger

**Windcrest®/McGraw-Hill**

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

FIRST EDITION
FIRST PRINTING



**Library of Congress Cataloging-in-Publication Data**

Dorfman, Len.
OS/2 Extra! : KBD, MOU, & VIO special functions revealed / edited by Len Dorfman and Marc J. Neuberger
p. cm.
Includes index.
ISBN 0-8306-4567-5 (pbk.)
1. Operating systems (Computers) 2. OS/2 (Computer file)
I. Neuberger, Marc J. II. Title.
QA76.76.063D673 1993
005.4'469—dc20 93-1498
CIP

Acquisitions Editor: Brad Schepp
Editorial Team: Robert E. Ostrander, Executive Editor
David M. McCandless, Book Editor
Production Team: Katherine G. Brown, Director
Book Team: Jaclyn J. Boone, Designer
Brian Allison, Associate Designer
Cover: Craig Reese, Hanover, Pa. WP1

*To Barbara and Rachel*
*for their loving kindness and spirited humor*
Len

*To Leah, Joe, and Miriam*
*for their love, support, and friendship*
Marc

*And to Gail Ostrow at IBM*
*for her enthusiastic support of this project*
Len & Marc

# *Contents*

## MOUSE (MOU) FUNCTIONS



## VIDEO (VIO) FUNCTIONS

# Preface

In part, this book came into being because Len hates using online documentation. He likes to read while sitting at his kitchen table sipping tea. He likes to read sitting in his living room listening to soft Baroque music. He does not like to read documentation while staring at his computer monitor.

Len needed some of the documentation presented in this book for his Instant OS/2! character-mode programming title. Len called IBM tech support. After a short waiting period, he became impatient and then called Marc for help.

Marc told Len about the .INF file IBM had released electronically in response to requests for VIO/KBD/MOU documentation. After Len began viewing the file, he reached for some aspirin.

Len begged Marc to write am .INF to ASCII conversion program so he could print out the VIO/KBD/MOU docs. With the help of a document written by Carl Hauser describing the .INF format, Marc took up the challenge and a few days later Len had his precious hardcopy of the VIO/KBD/MOU functions.

Len reasoned that he might not be the only person on the planet who prefers hardcopy to online documentation, so we decided to produce the book.

# Introduction

This book documents the OS/2 KBD/MOU/VIO subsystems. For whatever reason, IBM did not include documentation for the KBD/MOU/VIO calls in their OS/2 2.0 Technical Library. Although the current trend is toward GUI (Graphical User Interface) applications, there is still a place for full-screen character-mode programs. This book is designed to facilitate the creation (and perhaps more important, porting) of full-screen character-mode programs to run in OS/2 character-mode sessions.

The KBD/MOU/VIO subsystems provide a rich set of primitives for creating character-mode user interfaces. Unlike MS/DOS, OS/2 is a real operating system, and as such you generally need to go through the OS to get to the hardware. Where in MS/DOS you scribbled characters and attributes into addresses B800:0000-B800:07D0 (for example) to make characters appear on the screen, in OS/2 you must use the VIO calls. While you might lose some performance and flexibility, you gain enormously in terms of program reliability.

It's always fun to program the bare metal, but it's dangerous. Real operating systems like OS/2 provide a safety buffer between the programmer and the hardware.

This book is based on the .INF documentation provided by IBM. We have significantly reformatted the contents to suit the printed page. An important addition is the inclusion of `#define` constant names for bit masks and predefined values (where they are defined in the OS2 include files). In addition, we have added notes in some places where the actual behavior of the calls appears to deviate from the behavior documented by IBM.

We have chosen to orient the layout of the book to C programmers. Assembly language programmers are becoming a rare, though hardy, breed. We expect that they will be able to extract whatever information they need from the C call descriptions.

The book is organized into three parts. Part One describes the KBD interface. Part Two describes the MOU calls. Part Three describes the VIO system. The appendix gives a listing of the error codes returned by the KBD/MOU/VIO calls and their descriptions.

We hope that the information provided in this book will greatly ease the task of creating full screen character-mode programs that will run in an OS/2 full-screen session.

# Keyboard (KBD) functions

This part describes the OS/2 keyboard API interface.

## Notes

- Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
- Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
- Calls marked FAPI are present in the Family API.

**Table 1-1 Keyboard call list**

| Function call | Icon |
|---|---|
| KbdCharIn | FAPI xPM |
| KbdClose | xPM |
| KbdDeRegister | xWPM |
| KbdFlushBuffer | FAPI xPM |
| KbdFreeFocus | xPM |
| KbdGetCp | xPM |
| KbdGetFocus | xPM |
| KbdGetStatus | FAPI xPM |
| KbdOpen | xPM |
| KbdPeek | FAPI xPM |
| KbdRegister | xWPM |
| KbdSetCp | xPM |
| KbdSetCustXt | xPM |
| KbdSetFgnd | xPM |
| KbdSetStatus | FAPI xPM |
| KbdStringIn | FAPI xPM |
| KbdSynch | xWPM |
| KbdXlate | xPM |

# KbdCharIn (FAPI, xPM)

## Description

This call returns a character data record from the keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdCharIn(PKBDKEYINFO pkbci, USHORT fWait, HKBD hkbd);
```

`pkbci (PKBDKEYINFO)` - output
Address of the character data structure.

`chChar (UCHAR)`
ASCII character code. The scan code received from the keyboard is translated to the ASCII character code.

`chScan (UCHAR)`
Code received from the keyboard. The scan code received from the keyboard is translated to the ASCII character code.

`fbStatus (UCHAR)`
State of the keystroke event.

| Bit | Description |
|---|---|
| 7–6 | 00 = Undefined |
| | 01 = Final character, interim character flag off<br>(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& !(fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| 5 | 10 = Interim character<br>!(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& (fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| | 11 = Final character, interim character flag on.<br>(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& (fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| | 1 = Immediate conversion requested.<br>(fbStatus & KBDTRF_CONVERSION_REQUEST) |
| 4–2 | Reserved. |
| 1 | 0 = Scan code is a character. |
| | 1 = Scan code is not a character but is an extended key code from the keyboard. |
| 0 | 1 = Shift status returned without character.<br>(fbStatus & KBDTRF_SHIFT_KEY_IN) |

`bNlsShift (UCHAR)`
NLS shift status. Reserved, set to zero.

`fsState (USHORT)`
Shift key status.

| Bit | Description | |
|---|---|---|
| 15 | SysReq key down | (fsState & KBDSTF_SYSREQ) |
| 14 | CapsLock key down | (fsState & KBDSTF_CAPSLOCK) |
| 13 | NumLock key down | (fsState & KBDSTF_NUMLOCK) |
| 12 | ScrollLock key down | (fsState & KBDSTF_SCROLLLOCK) |
| 11 | Right Alt key down | (fsState & KBDSTF_RIGHTALT) |
| 10 | Right Ctrl key down | (fsState & KBDSTF_RIGHTCONTROL) |
| 9 | Left Alt key down | (fsState & KBDSTF_LEFTALT) |
| 8 | Left Ctrl key down | (fsState & KBDSTF_LEFTCONTROL) |
| 7 | Insert on | (fsState & KBDSTF_INSERT_ON) |
| 6 | CapsLock on | (fsState & KBDSTF_CAPSLOCK_ON) |
| 5 | NumLock on | (fsState & KBDSTF_NUMLOCK_ON) |
| 4 | ScrollLock on | (fsState & KBDSTF_SCROLL LOCK_ON) |
| 3 | Either Alt key down | (fsState & KBDSTF_ALT) |
| 2 | Either Ctrl key down | (fsState & KBDSTF_CONTROL) |
| 1 | Left Shift key down | (fsState & KBDSTF_LEFTSHIFT) |
| 0 | Right Shift key down | (fsState & KBDSTF_RIGHTSHIFT) |

`time (ULONG)`
Time stamp indicating when a key was pressed. It is specified in milliseconds from the time the system was started.

`fWait (USHORT)` - input
Wait if a character is not available.

| Value | Definition |
|---|---|
| 0 | Requestor waits for a character if one is not available. (fWait == IO_WAIT) |
| 1 | Requestor gets an immediate return if no character is available. (fWait == IO_NOWAIT) |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 375 | ERROR_KBD_INVALID_IOWAIT |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

On an enhanced keyboard, the secondary enter key returns the normal character 0DH and a scan code of E0H.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set with `KbdSetStatus`, the `CharData` record returned reflects changed shift information only.

Extended ASCII codes are identified with the status byte, bit 1 on and the ASCII character code being either 00H or E0H. Both conditions must be satisfied for the character to be an extended keystroke. For extended ASCII codes, the scan code byte returned is the second code (extended code). Usually the extended ASCII code is the scan code of the primary key that was pressed.

A thread in the foreground session that repeatedly polls the keyboard with `KbdCharIn` (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield to the CPU by issuing a `DosSleep` call for an interval of at least 5 milliseconds.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `KbdCharIn` when coding in the DOS mode:

- The `CharData` structure includes everything except the time stamp.
- Interim character is not supported
- Status can be 0 or 40H
- `KbdHandle` is ignored.

# KbdClose (xPM)

## Description

This call closes the existing logical keyboard identified by the keyboard handle.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdClose (HKBD hkbd);
```

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

`KbdClose` blocks while another thread has the keyboard focus (by way of `KbdGetFocus`) until the thread with the focus issues `KbdFreeFocus`. Therefore, to prevent `KbdClose` from blocking, it is recommended that `KbdClose` be issued only while the current thread has the focus. For example:

| | |
|---|---|
| `KbdGetFocus` | Wait until focus available on handle 0. |
| `KbdClose` | Close a logical keyboard handle. |
| `KbdClose` | Close another logical keyboard handle. |
| `KbdClose` | Close still another logical keyboard handle. |
| `KbdFreeFocus` | Give up the focus on handle 0. |

# KbdDeRegister (xWPM)

## Description

This call deregisters a keyboard subsystem previously registered within a session. Only the process that issued the `KbdRegister` may issue `KbdDeRegister`.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdDeRegister (void);
```

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 411 | ERROR_KBD_DEREGISTER |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

# KbdFlushBuffer (FAPI, xPM)

## Description

This call clears the keystroke buffer.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdFlushBuffer (HKBD hkbd);
```

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

`KbdFlushBuffer` completes when the handle has access to the physical keyboard (focus), or is equal to zero and no other handle has the focus.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. The `KbdHandle` is ignored when coding in the DOS mode.

# KbdFreeFocus (xPM)

## Description

This call frees the logical-to-physical keyboard bond created by `KbdGetFocus`.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdFreeFocus (HKBD hkbd);
```

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

0 NO_ERROR
439 ERROR_KBD_INVALID_HANDLE
445 ERROR_KBD_FOCUS_REQUIRED
464 ERROR_KBD_DETACHED
504 ERROR_KBD_EXTENDED_SG

## Remarks

`KbdFreeFocus` may be replaced by issuing `KbdRegister`. Unlike other keyboard subsystem functions, the replaced `KbdFreeFocus` is called only if there is an outstanding focus.

# KbdGetCp (xPM)

## Description

This call allows a process to query the code page being used to translate scan codes to ASCII characters.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdGetCp (ULONG ulReserved, PUSHORT pidCP, HKBD hkbd);
```

`ulReserved (ULONG)` - input
Reserved and must be set to zero.

`pidCP (PUSHORT)` - output
Address of the code page ID located in the application's data area. The keyboard support copies the current code page ID for a specified keyboard handle into this word. The code page ID is equivalent to one of the code page IDs specified in the CONFIG.SYS CODEPAGE = statement or 0000.

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 373 | ERROR_KBD_PARAMETER |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

The `CodePageID` is the currently active keyboard code page. A value of 0 indicates the code page translation table in use is the ROM code page translation table provided by the hardware.

# KbdGetFocus (xPM)

## Description

This call binds the logical keyboard to the physical keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdGetFocus (USHORT fWait, HKBD hkbd);
```

`fWait (USHORT)` - input
Wait if the physical keyboard is already in use by a logical keyboard.

| Value | Definition |
|---|---|
| 0 | Indicates that the caller wants to wait for the bond. (fWait == IO_WAIT) |
| 1 | Indicates that the caller does not want to wait for the bond. (fWait == IO_NOWAIT) |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 446 | ERROR_KBD_FOCUS_ALREADY_ACTIVE |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

The keyboard handle identifies which logical keyboard to bind to. If the physical keyboard is not bound to a logical or default keyboard, then the bind proceeds immediately. The logical keyboard, identified by the handle, receives all further keystrokes from the physical keyboard. If the physical keyboard is already in use by a logical keyboard, then the thread issuing `KbdGetFocus` waits until the bond can be made. Waiting threads do not execute in any definable order.

# KbdGetHWID (xPM)

## Description

Returns the attached keyboard's hardware-generated Identification value.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdGetHWID (PKBDHWID pkbdhwid, HKBD hkbd);
```

`pkbdhwid (PKBDHWID)` - input
: Pointer to the caller's data area where the following structure and data values are:

`cb (USHORT)` - input/output
: On input, this field should contain the length of the `KeyboardID` structure. The minimum input length value allowed is 2. On output, this field contains the actual number of bytes returned.

`idKbd (USHORT)` - output
: OS/2 supported keyboards and their hardware generated IDs are as follows:

| ID | Keyboard |
|---|---|
| 0000H | Undetermined keyboard type |
| 0001H | PC-AT standard keyboard |
| AB41H | 101-key enhanced keyboard |
| AB41H | 102-key enhanced keyboard |
| AB54H | 88- and 89-key enhanced keyboards |
| AB85H | 122-key enhanced keyboard |

`usReserved1 (USHORT)`
: Reserved and returned set to zero.

`usReserved2 (USHORT)`
: Reserved and returned set to zero.

`hkbd (HKBD)` - input
: Word identifying the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 373 | ERROR_KBD_PARAMETER |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

In past OS/2 releases, all keyboards could be supported by knowing the hardware family information available with keyboard IOCTL 77H. However, with the addition of the 122-key keyboard, recognition was not containable by hardware family information alone. The 122-key keyboard has a number of differences from other keyboards. Therefore, applications performing keystroke specific functions might need to determine specifically which keyboard is attached.

This function is of particular usefulness for applications providing Custom Translate Tables and mapping keyboard layouts.

# KbdGetStatus (FAPI, xPM)

## Description

This call gets the current state of the keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdGetStatus (PKBDINFO pkbdinfo, HKBD hdbd);
```

`pkbdinfo (PKBDINFO)` - output
Address of the keyboard status structure:

`cb (USHORT)`
Length, in bytes, of this data structure, including length.

10 Only valid value.

`fsMask (USHORT)`
State as follows:

| Bit | Description |
|---|---|
| 15–9 | Reserved, set to zero. |
| 8 | Shift return is on.<br>(fsMask & KEYBOARD_SHIFT_REPORT) |
| 7 | Length of the turnaround character (meaningful only if bit 6 is on).<br>(fsMask & KEYBOARD_2B_TURNAROUND) |
| 6 | Turnaround character is modified.<br>(fsMask & KEYBOARD_MODIFY_TURNAROUND) |
| 5 | Interim character flags are modified.<br>(fsMask & KEYBOARD_MODIFY_INTERIM) |
| 4 | Shift state is modified.<br>(fsMask & KEYBOARD_MODIFY_STATE) |
| 3 | ASCII mode is on.<br>(fsMask & KEYBOARD_ASCII_MODE) |
| 2 | Binary mode is on.<br>(fsMask & KEYBOARD_BINARY_MODE) |
| 1 | Echo off.<br>(fsMask & ECHO_OFF) |
| 0 | Echo on.<br>(fsMask & ECHO_ON) |

`chTurnAround (USHORT)`
Definition of the turnaround character. In ASCII and extended-

ASCII format, the turnaround character is defined as the carriage return. In ASCII format only, the turnaround character is defined in the low-order byte.

`fsInterim (USHORT)`
Interim character flags:

| Bit | Description |
|---|---|
| 15–8 | NLS shift state. |
| 7 | Interim character flag is on. |
| 6 | Reserved, set to zero. |
| 5 | Application requested immediate conversion. |
| 4–0 | Reserved, set to zero. |

`fsState (USHORT)`
Shift state as follows:

| Bit | Description | |
|---|---|---|
| 15 | SysReq key down | (fsState & KBDSTF_SYSREQ) |
| 14 | CapsLock key down | (fsState & KBDSTF_CAPSLOCK) |
| 13 | NumLock key down | (fsState & KBDSTF_NUMLOCK) |
| 12 | ScrollLock key down | (fsState & KBDSTF_SCROLLLOCK) |
| 11 | Right Alt key down | (fsState & KBDSTF_RIGHTALT) |
| 10 | Right Ctrl key down | (fsState & KBDSTF_RIGHTCONTROL) |
| 9 | Left Alt key down | (fsState & KBDSTF_LEFTALT) |
| 8 | Left Ctrl key down | (fsState & KBDSTF_LEFTCONTROL) |
| 7 | Insert on | (fsState & KBDSTF_INSERT_ON) |
| 6 | CapsLock on | (fsState & KBDSTF_CAPSLOCK_ON) |
| 5 | NumLock on | (fsState & KBDSTF_NUMLOCK_ON) |
| 4 | ScrollLock on | (fsState & KBDSTF_SCROLLLOCK_ON) |
| 3 | Either Alt key down | (fsState & KBDSTF_ALT) |
| 2 | Either Ctrl key down | (fsState & KBDSTF_CONTROL) |
| 1 | Left Shift key down | (fsState & KBDSTF_LEFTSHIFT) |
| 0 | Right Shift key down | (fsState & KBDSTF_RIGHTSHIFT) |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 376 | ERROR_KBD_INVALID_LENGTH |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

The initial state of the keyboard is established by the system at application load time. Some default states may be modified by the application through `KbdSetStatus`. `KbdGetStatus` returns only those keyboard parameters initially set by `KbdSetStatus`. The returned parameters are

- Input mode
- Interim character flags
- Shift state
- Echo state
- Turnaround character

`KbdGetStatus` completes only when the handle has access to the physical keyboard (focus) or the handle is 0 and no other handle has the focus.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `KbdGetStatus` when coding in the DOS mode:

- Interim character is not supported
- Turnaround character is not supported
- `NLS_SHIFT_STATE` is always NULL.
- `KbdHandle` is ignored.

# KbdOpen (xPM)

## Description

This call creates a new logical keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdOpen (PHKBD phkbd);
```

`phkbd (PHKBD)` - output
Address of the logical keyboard.

## Return Value

| | |
|---|---|
| 0 | NO_ERROR |
| 440 | ERROR_KBD_NO_MORE_HANDLE |
| 441 | ERROR_KBD_CANNOT_CREATE_KCB |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

`KbdOpen` blocks while another thread has the keyboard focus (by way of `KbdGetFocus`) until the thread with the focus issues `KbdFreeFocus`. Therefore, to prevent `KbdOpen` from blocking, it is recommended that `KbdOpen` be issued only while the current thread has the focus. For example:

| | |
|---|---|
| `KbdGetFocus` | Wait until focus available on handle 0 |
| `KbdOpen` | Get a logical keyboard handle |
| `KbdOpen` | Get another logical keyboard handle |
| `KbdOpen` | Get yet another logical keyboard handle |
| `KbdFreeFocus` | Give up the focus on handle 0. |

# KbdPeek (FAPI, xPM)

## Description

This call returns any available character data record from the keyboard without removing it from the buffer.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdPeek (PKBDKEYINFO pkbci, HKBD hkbd);
```

`pkbci (PKBDKEYINFO)` - output
Address of the character data information:

`chChar (UCHAR)`
ASCII character code. The scan code received from the keyboard is translated to the ASCII character code.

`chScan (UCHAR)`
Code received from the keyboard hardware.

`fbStatus (UCHAR)`
State of the keystroke event:

| Bit | | Description |
|---|---|---|
| 7–6 | 00 = | Undefined |
| | 01 = | Final character, interim character flag off<br>(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& !(fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| | 10 = | Interim character<br>!(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& (fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| | 11 = | Final character, interim character flag on.<br>(fbStatus & KBDTRF_FINAL_CHAR_IN)<br>&& (fbStatus & KBDTRF_INTERIM_CHAR_IN) |
| 5 | 1 = | Immediate conversion requested.<br>(fbStatus & KBDTRF_CONVERSION_REQUEST) |
| 4–2 | Reserved. | |
| 1 | 0 = | Scan code is a character. |
| | 1 = | Scan code is not a character; is an extended key code from the keyboard. |
| 0 | 1 = | Shift status returned without character.<br>(fbStatus & KBDTRF_SHIFT_KEY_IN) |

`bNlsShift (UCHAR)`
NLS shift status. Reserved, set to zero.

`fsState (USHORT)`
Shift key status.

| Bit | Description | |
|---|---|---|
| 15 | SysReq key down | (fsState & KBDSTF_SYSREQ) |
| 14 | CapsLock key down | (fsState & KBDSTF_CAPSLOCK) |
| 13 | NumLock key down | (fsState & KBDSTF_NUMLOCK) |
| 12 | ScrollLock key down | (fsState & KBDSTF_SCROLLLOCK) |
| 11 | Right Alt key down | (fsState & KBDSTF_RIGHTALT) |
| 10 | Right Ctrl key down | (fsState & KBDSTF_RIGHTCONTROL) |
| 9 | Left Alt key down | (fsState & KBDSTF_LEFTALT) |
| 8 | Left Ctrl key down | (fsState & KBDSTF_LEFTCONTROL) |
| 7 | Insert on | (fsState & KBDSTF_INSERT_ON) |
| 6 | CapsLock on | (fsState & KBDSTF_CAPSLOCK_ON) |
| 5 | NumLock on | (fsState & KBDSTF_NUMLOCK_ON) |
| 4 | ScrollLock on | (fsState & KBDSTF_SCROLLLOCK_ON) |
| 3 | Either Alt key down | (fsState & KBDSTF_ALT) |
| 2 | Either Ctrl key down | (fsState & KBDSTF_CONTROL) |
| 1 | Left Shift key down | (fsState & KBDSTF_LEFTSHIFT) |
| 0 | Right Shift key down | (fsState & KBDSTF_RIGHTSHIFT) |

`time (ULONG)`
Time stamp indicating when a key was pressed. It is specified in milliseconds from the time the system was started.

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

On an enhanced keyboard, the secondary enter key returns the normal character 0DH and a scan code of E0H.

Double-byte character codes (DBCS) require two function calls to obtain the entire code.

If shift report is set with `KbdSetStatus` the `CharData` record returned, reflects changed shift information only.

Extended ASCII codes are identified with the status byte, bit 1 on and the ASCII character code being either 00H or E0H. Both conditions must be satisfied for the character to be an extended keystroke. For extended ASCII codes, the scan code byte returned is the second code (extended code). Usually the extended ASCII code is the scan code of the primary key that was pressed.

A thread in the foreground session that repeatedly polls the keyboard with KbdCharIn (with no wait), can prevent all regular priority class threads from executing. If polling must be used and a minimal amount of other processing is being performed, the thread should periodically yield the CPU by issuing a `DosSleep` call for an interval of at least 5 milliseconds.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `KbdPeek` when coding for the DOS mode:

- The `CharData` structure includes everything except the time stamp.
- Interim character is not supported.
- Status can be 0 or 1.
- `KbdHandle` is ignored.

# KbdRegister (xWPM)

## Description

This call registers a keyboard subsystem within a session.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdRegister (PSZ pszModName, PSZ pszEntryPt, ULONG FunMask);
```

`pszModName (PSZ)` - input
Address of the dynamic link module name. Maximum length is 9 bytes (including ASCIIZ terminator).

`pszEntryPt (PSZ)` - input
Address of the dynamic link entry point name of a routine that receives control when any of the registered functions are called. Maximum length is 33 bytes (including ASCIIZ terminator).

`FunMask (ULONG)` - input
A bit mask where each bit identifies a keyboard function being registered. The bit values are

| Bit | Registered function | Mask |
|---|---|---|
| 31–15 | Reserved, must be set to zero. | |
| 14 | KbdGetHWId | |
| 13 | KbdSetCustXt | (FunMask & KR_KBDSETCUSTXT) |
| 12 | KbdXlate | (FunMask & KR_KBDXLATE) |
| 11 | KbdSetCp | (FunMask & KR_KBDSETCP) |
| 10 | KbdGetCp | (FunMask & KR_KBDGETCP) |
| 9 | KbdFreeFocus | (FunMask & KR_KBDFREEFOCUS) |
| 8 | KbdGetFocus | (FunMask & KR_GETFOCUS) |
| 7 | KbdClose | (FunMask & KR_KBDCLOSE) |
| 6 | KbdOpen | (FunMask & KR_KBDOPEN) |
| 5 | KbdStringIn | (FunMask & KR_KBDSTRINGIN) |
| 4 | KbdSetStatus | (FunMask & KR_KBDSETSTATUS) |
| 3 | KbdGetStatus | (FunMask & KR_KBDGETSTATUS) |
| 2 | KbdFlushBuffer | (FunMask & KR_KBDFLUSHBUFFER) |
| 1 | KbdPeek | (FunMask & KR_KBDPEEK) |
| 0 | KbdCharIn | (FunMask & KR_KBDCHARIN) |

## Return value

0 NO_ERROR
408 ERROR_KBD_INVALID_ASCIIZ

409 ERROR_KBD_INVALID_MASK
410 ERROR_KBD_REGISTER
464 ERROR_KBD_DETACHED
504 ERROR_KBD_EXTENDED_SG

## Remarks

There can be only one `KbdRegister` call outstanding for each session without an intervening `KbdDeRegister`. `KbdDeRegister` must be issued by the same process that issued the `KbdRegister`.

## Description

This call allows the process to set the code page used to translate keystrokes received from the keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdSetCp (USHORT usReserved, USHORT pidCP, HKBD hkbd);
```

`usReserved (USHORT)` - input
Reserved and must be set to zero.

`pidCP (USHORT)` - input
CodePageID represents a code-page ID in the application's data area. The code-page ID must be equivalent to one of the code-page IDs specified on the CONFIG.SYS CODEPAGE = statement or 0000. If the code-page ID does not match one of the IDs on the CODEPAGE = statement, an error results. The code-page word must have one of these code-page identifiers:

| Identifier | Description |
|---|---|
| 0 | Resident code page |
| 437 | IBM PC US 437 |
| 850 | Multilingual |
| 860 | Portuguese |
| 863 | Canadian-French |
| 865 | Nordic |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 448 | ERROR_KBD_INVALID_CODEPAGE |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

Keyboard code page support is not available without the `DEVINFO=KBD` statement in the CONFIG.SYS file. Refer to *IBM Operating System/2.0 Command Reference* for a complete description of CODEPAGE and DEVINFO.

## Description

This call installs, on the specified handle, the translate table that this call points to. This translate table affects only this handle.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdSetCustXt (PUSHORT usCodePage, HKBD hkbd);
```

`usCodePage (PUSHORT)` - input
A pointer to the translation table used to translate scan code to ASCII code for a specified handle. The format of the translate table is documented in the Set Code Page IOCTL 50H. Refer to *IBM Operating System/2.0 Technical Library Physical Drive Device Drivers Reference* for a complete discussion of Set Code Page IOCTL 50H.

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 377 | ERROR_KBD_INVALID_ECHO_MASK |
| 378 | ERROR_KBD_INVALID_INPUT_MASK |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

The translate table must be maintained in the caller's memory. No copy of the translate table is made by `KbdSetCustXt`. `KbdSetCp` reverses the action of `KbdSetCustXt` and sets the handle equal to one of the system translate tables. If memory is dynamically allocated by the caller for the translate table and is freed before the `KbdSetCp` is performed, `KbdSetCp` and future translations might fail.

# KbdSetFgnd (xPM)

## Description

This call raises the priority of the foreground keyboard's thread.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdSetFgnd(void);
```

## Return Value

| | |
|---|---|
| 0 | NO_ERROR |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

`KbdSetFgnd` marks the current process that owns the keyboard. Threads in this process receive a priority boost. The previous foreground keyboard threads lose their priority boost. This function should only be issued by a Keyboard Subsystem during `KbdCharIn` or `KbdStringIn` processing.

## Description

This call sets the characteristics of the keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdSetStatus (PKBDINFO pkbdinfo, HKBD hkbd);
```

`pkbdinfo (PKBDINFO)` - **input**
Address of the keyboard status structure:

`cb (USHORT)`
Length, in bytes, of this data structure, including length.

10 Only valid value.

`fsMask (USHORT)`
The system state altered by this call. If bits 0 and 1 are off, the echo state of the system is not altered. If bits 2 and 3 are off, the binary and ASCII state of the system is not altered. If bits 0 and 1 are on, or if bits 2 and 3 are on, the function returns an error. If binary mode is set, echo is ignored.

| Bit | Description |
|---|---|
| 15–9 | Reserved, set to zero. |
| 8 | Shift return is on.<br>(fsMask & KEYBOARD_SHIFT_REPORT) |
| 7 | Length of the turnaround character (meaningful only if bit 6 is on).<br>(fsMask & KEYBOARD_2B_TURNAROUND) |
| 6 | Turnaround character is modified.<br>(fsMask & KEYBOARD_MODIFY_TURNAROUND) |
| 5 | Interim character flags are modified.<br>(fsMask & KEYBOARD_MODIFY_INTERIM) |
| 4 | Shift state is modified.<br>(fsMask & KEYBOARD_MODIFY_STATE) |
| 3 | ASCII mode is on.<br>(fsMask & KEYBOARD_ASCII_MODE) |
| 2 | Binary mode is on.<br>(fsMask & KEYBOARD_BINARY_MODE) |
| 1 | Echo off.<br>(fsMask & ECHO_OFF) |
| 0 | Echo on.<br>(fsMask & ECHO_ON) |

`chTurnAround (USHORT)`
Definition of the turnaround character. In ASCII and extended-ASCII format, the turnaround character is defined as the carriage return. In ASCII format only, the turnaround character is defined in the low-order byte.

`fsInterim (USHORT)`
Interim character flags:

| Bit | Description |
|---|---|
| 15–8 | NLS shift state. |
| 7 | Interim character flag is on |
| 6 | Reserved, set to zero |
| 5 | Application requested immediate conversion |
| 4–0 | Reserved, set to zero |

`fsState (USHORT)`
Shift state.

| Bit | Description | |
|---|---|---|
| 15 | SysReq key down | (fsState & KBDSTF_SYSREQ) |
| 14 | CapsLock key down | (fsState & KBDSTF_CAPSLOCK) |
| 13 | NumLock key down | (fsState & KBDSTF_NUMLOCK) |
| 12 | ScrollLock key down | (fsState & KBDSTF_SCROLLLOCK) |
| 11 | Right Alt key down | (fsState & KBDSTF_RIGHTALT) |
| 10 | Right Ctrl key down | (fsState & KBDSTF_RIGHTCONTROL) |
| 9 | Left Alt key down | (fsState & KBDSTF_LEFTALT) |
| 8 | Left Ctrl key down | (fsState & KBDSTF_LEFTCONTROL) |
| 7 | Insert on | (fsState & KBDSTF_INSERT_ON) |
| 6 | CapsLock on | (fsState & KBDSTF_CAPSLOCK_ON) |
| 5 | NumLock on | (fsState & KBDSTF_NUMLOCK_ON) |
| 4 | ScrollLock on | (fsState & KBDSTF_SCROLLLOCK_ON) |
| 3 | Either Alt key down | (fsState & KBDSTF_ALT) |
| 2 | Either Ctrl key down | (fsState & KBDSTF_CONTROL) |
| 1 | Left Shift key down | (fsState & KBDSTF_LEFTSHIFT) |
| 0 | Right Shift key down | (fsState & KBDSTF_RIGHTSHIFT) |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

0 NO_ERROR
376 ERROR_KBD_INVALID_LENGTH
377 ERROR_KBD_INVALID_ECHO_MASK
378 ERROR_KBD_INVALID_INPUT_MASK
439 ERROR_KBD_INVALID_HANDLE
445 ERROR_KBD_FOCUS_REQUIRED

447 ERROR_KBD_KEYBOARD_BUSY
464 ERROR_KBD_DETACHED
504 ERROR_KBD_EXTENDED_SG

## Remarks

Shift return (bit 8 in sysstate) must be disabled in ASCII mode. `KbdSetStatus` is ignored for a Vio-windowed application.

## Family API Considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `KbdSetStatus` when coding in the DOS mode:

- `KbdSetStatus` does not accept a bit mask of 10 (ASCII on, Echo Off).
- Raw (binary) Mode and Echo On are not supported and return an error if requested.
- `KbdHandle` is ignored.
- Interim character is not supported.
- Turnaround character is not supported.

# KbdStringIn (FAPI, xPM)

## Description

This call reads a character string (character codes only) from the keyboard.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdStringIn (PCH pch, PSTRINGINBUF pchIn,
                    USHORT fsWait, HKBD hkbd);
```

`pch (PCH)` - output
Address of the character string buffer.

`pchIn (PSTRINGINBUF)` - input/output
Address of the length of the character string buffer. On entry, `buflen` is the maximum length, in bytes, of the buffer. The maximum length that can be specified is 255. Template processing has meaning only in the ASCII mode.

`cb (USHORT)`
Length of the input buffer.

`cchIn (USHORT)`
Number of bytes read into the buffer.

`fsWait (USHORT)` - input
Wait if a character is not available.

| Value | Definition |
|---|---|
| 0 | Wait. In Binary input mode, the requestor waits until CharBuffer is full. In ASCII input mode, the requestor waits until a carriage return is pressed.<br>(fsWait == IO_WAIT) |
| 1 | No wait. The requestor gets an immediate return if no characters are available. If characters are available, `KbdStringIn` returns immediately with as many characters as are available (up to the maximum). No wait is not supported in ASCII input mode.<br>(fsWait == IO_NOWAIT) |

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

```
  0 NO_ERROR
375 ERROR_KBD_INVALID_IOWAIT
439 ERROR_KBD_INVALID_HANDLE
445 ERROR_KBD_FOCUS_REQUIRED
464 ERROR_KBD_DETACHED
504 ERROR_KBD_EXTENDED_SG
```

## Remarks

The character strings may be optionally echoed on the display if echo mode is set. When echo is on each character is echoed as it is read from the keyboard. Echo mode and BINARY mode are mutually exclusive. Reference `KbdSetStatus` and `KbdGetStatus` for more information.

The default input mode is ASCII. In ASCII mode, 2-byte character codes only return in complete form. An extended ASCII code is returned in a 2-byte string. The first byte is 0DH or E0H and the next byte is an extended code.

In input mode (BINARY, ASCII), the following returns can be set and retrieved with `KbdSetStatus` and `KbdGetStatus`:

- Turnaround character
- Echo mode
- Interim character flag
- Shift state

The received input length is also used by the `KbdStringIn` line edit functions for re-displaying and entering a caller specified string. On the next `KbdStringIn` call the received input length indicates the length of the input buffer that may be recalled by the user using the line editing keys. A value of 0 inhibits the line editing function for the current `KbdStringIn` request.

`KbdStringIn` completes when the handle has access to the physical keyboard (focus), or is equal to zero and no other handle has the focus.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restrictions apply to `KbdStringIn` when coding in the DOS mode:

- `KbdHandle` is ignored

Refer to the `DosRead`'s "Family API considerations" for differences between DOS and OS/2 mode when reading from a handle opened to the CON device.

# KbdSynch (xWPM)

## Description

This call synchronizes access from a keyboard subsystem to the keyboard device driver.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdSynch (USHORT fsWait);
```

`fsWait (USHORT)` - **input**
Wait for the bond. Values are:

| Value | Definition |
|---|---|
| 0 | Indicates the requestor does not wait for access to the device driver.<br>(fsWait == IO_WAIT) |
| 1 | Indicates the requestor waits for access to the device driver.<br>(fsWait == IO_NOWAIT) |

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 121 | ERROR_SEM_TIMEOUT |

## Remarks

`KbdSynch` blocks all other threads within a session until return from the subsystem to the router. To ensure proper synchronization, `KbdSynch` should be issued by a keyboard subsystem if it intends to issue a `DosDevIOCtl` or access dynamically shared data. `KbdSynch` does not protect globally shared data from threads in other sessions.

## Description

This call translates scan codes with shift states into ASCII codes.

```
#define INCL_KBD
#include <os2.h>

USHORT KbdXlate (PKBDTRANS pkbdtrans, HKBD hkbd);
```

`pkbdtrans (PKBDTRANS)` - input
Address of the translation record structure: `chChar`, `chScan`, `fbStatus`, `bNlsShift`, `fsState`, time (KBDKEYINFO) character data information structure as defined in `KbdCharIn` call.

`fsDD (USHORT)`
See the `KbdDDFlagWord` call in the "Physical keyboard device driver" section of IBM OS/2 2.0 Technical Library Physical Device Driver Reference.

`fsXlate (USHORT)`
Translation flag:

| Value | Definition |
|---|---|
| 0 | Translation incomplete. |
| 1 | Translation complete. |

`fsShift (USHORT)`
Identifies the state of translation across successive calls; initially the value should be zero. It might take several calls to this function to complete a character. The value should not be changed unless a new translation is required (that is, reset value to zero).

`sZero (USHORT)`
See description for `fsShift`.

`hkbd (HKBD)` - input
Default keyboard or the logical keyboard.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 439 | ERROR_KBD_INVALID_HANDLE |
| 445 | ERROR_KBD_FOCUS_REQUIRED |
| 447 | ERROR_KBD_KEYBOARD_BUSY |
| 464 | ERROR_KBD_DETACHED |
| 504 | ERROR_KBD_EXTENDED_SG |

## Remarks

It might take several calls to complete a translation because of accent key combinations, or other complex operations.

The `fsShift` and `sZero` are for use by the keyboard translation routines. These fields are reserved and must only be accessed by the caller prior to starting a translation sequence and then they must be set to zero. The `KbdXlate` function is intended to be used for translating a particular scan code for a given shift state. The `KbdXlate` function is not intended to be a replacement for the OS/2 system keystroke translation function.

# Mouse (MOU) functions

This part describes the OS/2 Mouse API interface.

For information regarding mouse device drivers, mouse pointer draw device, mouse installation, and mouse IOCTLs, refer to *IBM OS/2 2.0 Technical Library Physical Device Drivers Reference.*

## Notes

- Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
- Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
- Calls marked FAPI are present in the Family API.

**Table 2-1 Mouse call list**

| Function call | Icon |
|---|---|
| MouClose | xPM |
| MouDeRegister | xWPM |
| MouDrawPtr | xPM |
| MouFlushQue | xPM |
| MouGetDevStatus | xPM |
| MouGetEventMask | xPM |
| MouGetNumButtons | xPM |
| MouGetNumMickeys | xPM |
| MouGetNumQueEl | xPM |
| MouGetPtrPos | xPM |
| MouGetPtrShape | xPM |
| MouGetScaleFact | xPM |
| MouInitReal | xWPM |
| MouOpen | xPM |
| MouReadEventQue | xPM |
| MouRegister | xWPM |
| MouRemovePtr | xPM |
| MouSetDevStatus | xPM |
| MouSetEventMask | xPM |
| MouSetPtrPos | xPM |
| MouSetPtrShape | xPM |
| MouSetScaleFact | xPM |
| MouSynch | xWPM |

# MouClose (xPM)

## Description

This call closes the mouse device for the current session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouClose (HMOU hmou);
```

`hmou (HMOU)` - input
Mouse device handle from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouClose` closes the mouse device for the current session and removes the mouse device driver handle from the list of valid open mouse device handles.

# MouDeRegister (xWPM)

## Description

This call deregisters a mouse subsystem previously registered within a session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouDeRegister (void);
```

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 416 | ERROR_MOUSE_DEREGISTER |
| 466 | ERROR_MOU_DETACHED |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

Processes issuing `MouDeRegister` calls must conform to the following rules:

- The process that issued the `MouRegister` must release the session (by a `MouDeRegister`) from the registered subsystem before another PID may issue `MouRegister`.
- The process that issued the `MouRegister` is the only process that may issue `MouDeRegister` against the currently registered subsystem.
- After the owning process has released the subsystem with a `MouDeRegister`, any other process in the session may issue a `MouRegister` and therefore modify the mouse support for the entire session.

# MouDrawPtr (xPM)

## Description

This call allows a process to notify the mouse device driver that an area previously restricted to the pointer image is now available to the mouse device driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouDrawPtr (HMOU hmou);
```

`hmou (HMOU)` - input
Mouse device handle from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

The collision area (the pointer image restricted area) is established by `MouOpen` and by `MouRemovePtr`. `MouDrawPtr` nullifies the effect of the `MouRemovePtr` command. If there was no previous `MouDrawPtr` command or if a previous `MouDrawPtr` command has already nullified the collision area, the `MouRemovePtr` command is effectively a null operation.

This call is required to begin session pointer image drawing. Immediately after `MouOpen` is issued, the collision area is defined as the size of the display. A `MouDrawPtr` is issued to begin pointer drawing after the `MouOpen`.

# MouFlushQue (xPM)

## Description

This call directs the mouse driver to flush (empty) the mouse event queue and the monitor chain data for the session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouFlushQue (HMOU hmou);
```

`hmou (HMOU)` - **input**
Mouse device handle from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

# MouGetDevStatus (xPM)

## Description

This call returns status flags for the installed mouse device driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetDevStatus (PUSHORT pfsDevStatus, HMOU hmou);
```

`pfsDevStatus (PUSHORT)` - output

Address of the current status flag settings for the installed mouse device driver. The return value is a 2-byte set of bit flags.

| Bit | Description |
|---|---|
| 15–10 | Reserved, set to zero. |
| 9 | Set if mouse data returned in mickeys, not pels.<br>(*pfsDevStatus & MOUSE_MICKEYS) |
| 8 | Set if the drawing operations for pointer draw routine are disabled.<br>(*pfsDevStatus & MOUSE_DISABLED) |
| 7–4 | Reserved, set to zero. |
| 3 | Set if pointer draw routine disabled by unsupported mode.<br>(*pfsDevStatus & MOUSE_UNSUPPORTED_MODE) |
| 2 | Set if flush in progress.<br>(*pfsDevStatus & MOUSE_FLUSH) |
| 1 | Set if block read in progress.<br>(*pfsDevStatus & MOUSE_BLOCKREAD) |
| 0 | Set if event queue busy with I/O.<br>(*pfsDevStatus & MOUSE_QUEUEBUSY) |

`hmou (HMOU)` - input

Mouse device handle from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

# MouGetEventMask (xPM)

## Description

This call returns the current value of the mouse event queue mask.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetEventMask (PUSHORT pfsEvents, HMOU hmou);
```

`pfsEvents (PUSHORT)` - output

Address in application storage where the current mouse device driver's event mask is returned to the caller by the mouse device driver.

The `EventMask` is set by `MouSetEventMask` and has the following definition:

| Bit | Description |
|---|---|
| 15–7 | Reserved, set to zero. |
| 6 | Set to report button 3 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN3_DOWN) |
| 5 | Set to report button 3 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN3_DOWN) |
| 4 | Set to report button 2 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN2_DOWN) |
| 3 | Set to report button 2 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN2_DOWN) |
| 2 | Set to report button 1 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN1_DOWN) |
| 1 | Set to report button 1 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN1_DOWN) |
| 0 | Set to report mouse motion events with no button press/release events.<br>(*pfsEvents & MOUSE_MOTION) |

`hmou (HMOU)` - input

Handle of the mouse device from a previous `MouOpen`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |

501 ERROR_MOUSE_NO_CONSOLE
505 ERROR_MOU_EXTENDED_SG

## Remarks

Buttons are logically numbered from left to right unless the mouse has been configured as left-handed, in which case the buttons are numbered from right to left.

# MouGetNumButtons (xPM)

## Description

This call returns the number of buttons supported on the installed mouse driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetNumButtons (PUSHORT pcButtons, HMOU hmou);
```

`pcButtons (PUSHORT)` - output
Address of the number of physical buttons. The return values for the number of buttons supported are:

| Value | Definition |
|---|---|
| 1 | One mouse button |
| 2 | Two mouse buttons |
| 3 | Three mouse buttons. |

`hmou (HMOU)` - input
Handle of the mouse device from a previous `MouOpen`.

## Return value

385 ERROR_MOUSE_NO_DEVICE
466 ERROR_MOU_DETACHED
501 ERROR_MOUSE_NO_CONSOLE
505 ERROR_MOU_EXTENDED_SG

# MouGetNumMickeys (xPM)

## Description

This call returns the number of mickeys in each centimeter for the installed mouse driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetNumMickeys (PUSHORT pcMickeys, HMOU hmou);
```

`pcMickeys (PUSHORT)` - output
: Address of the number of physical mouse motion units. Mouse motion units are reported in mickeys in each centimeter. This value is constant based upon the mouse device attached.

`hmou (HMOU)` - input
: Handle of the mouse device from a previous `MouOpen`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

# MouGetNumQueEl (xPM)

## Description

This call returns the current status for the mouse device driver event queue.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetNumQueEl (PMOUQUEINFO qmouqi, HMOU hmou);
```

`qmouqi (PMOUQUEINFO)` - output

Address of the mouse queue status structure:

`cEvents (USHORT)`

Current number of event queue elements, in the range 0 <> value <> maxnumqelements.

`cMaxEvents (USHORT)`

Maximum number of queue elements as specified in the `QSIZE = NN` parameter in `DEVICE=MOUSExxx.SYS` statement in CONFIG.SYS.

`hmou (HMOU)` - input

Contains the handle of the mouse device obtained from a previous `MouOpen`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

The maxnumqelements returned by this function is established during mouse device driver configuration. See the mouse `DEVICE=MOUSExxx.SYS` statement in the *IBM OS/2 2.0 Command Reference* for further details.

## Description

This call queries the mouse driver to determine the current row and column coordinate position of the mouse pointer.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetPtrPos (PPTRLOC pmouLoc, HMOU hmou);
```

`pMouLoc (PPTRLOC)` - output
Address of the mouse pointer position structure:

`row (USHORT)`
Current pointer row coordinate (pels or characters).

`col (USHORT)`
Current pointer column coordinate (pels or characters).

`hmou (HMOU)` - input
Contains the handle of the mouse device obtained from a previous `MouOpen`.

## Return value

0 NO_ERROR
385 ERROR_MOUSE_NO_DEVICE
466 ERROR_MOU_DETACHED
501 ERROR_MOUSE_NO_CONSOLE
505 ERROR_MOU_EXTENDED_SG

## Remarks

For a text window (VIO) application, the text window is a view on the larger logical video buffer (LVB). The mouse pointer can be outside that view and still be within the extent of the LVB. `MouGetPtrPos` then returns the coordinates of the cell under the mouse pointer. If the pointer is outside the LVB image extent, the coordinates of the nearest LVB cell are returned. In either case, the LVB is scrolled until the reported LVB cell appears within the view window.

# MouGetPtrShape (xPM)

## Description

This call allows a process to get (copy) the pointer shape for the session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetPtrShape (PBYTE pBuf, PPTRSHAPE pmoupsInfo, HMOU hmou);
```

`pBuf (PBYTE)` - output

Address of an area in application storage where the pointer draw device driver returns the pointer bit image. See `MouSetPtrShape` for a further description of the resulting content of this buffer.

`pmoupsInfo (PPTRSHAPE)` - input/output

Address of a structure in application storage where the application stores the data necessary for the pointer device driver to return information about the Row by Col image for each bit plane for the mode the display is currently running. See `MouSetPtrShape` for a further description of the contents of this structure.

`cb (USHORT)`

Length of the pointer buffer available for the pointer device driver to build a Row by Col image for each bit plane for the mode the display is currently running. This value is supplied by the application. If the value is too small, pointer draw places the true length of the image in this field, and returns an error.

For all OS/2 system-supported modes, `TotLength` is specified in bytes and is equal to:

**Mono & text modes**

For text mode height and width must be 1, so length is always 4.

cb = (height in chars) * (width in chars) * 2 * 2
= 1 * 1 * 2 * 2
= 4

**Graphics mode**

Width-in-pels must be a multiple of 8.

cb = (height in pels)*(width in pels)*(bits per pel)*2/8

**Modes 4 and 5 (320 × 200)**

cb = (height) * (width) * 2 * 2 / 8

**Mode 6 (640 × 200)**

cb = (height) * (width) * 1 * 2 / 8

Length calculations produce byte boundary buffer sizes.

`col (USHORT)`
Number of columns in the mouse shape. In graphics modes, this field contains the pel width (columns) of the mouse shape for the session and must be greater than or equal to 1. In text modes, col must equal 1.

`row (USHORT)`
Number of rows in the mouse shape. In graphics modes, this field contains the pel height (rows) of the mouse shape for the session and must be greater than or equal to 1. In text modes, row must equal 1.

`colHot (USHORT)`
This value is returned by the mouse device driver to indicate the relative column offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

`rowHot (USHORT)`
This value is returned by the mouse device driver to indicate the relative row offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

`hmou (HMOU)` - input
Handle of the mouse device from a previous MouOpen.

## Return value

0 NO_ERROR
385 ERROR_MOUSE_NO_DEVICE
387 ERROR_MOUSE_INV_PARMS
466 ERROR_MOU_DETACHED
501 ERROR_MOUSE_NO_CONSOLE
505 ERROR_MOU_EXTENDED_SG

## Remarks

The application passes a parameter list with the same meaning as defined for `MouSetPtrShape` to the mouse device driver. The mouse device driver copies the parameters that describe the pointer shape and attributes into

the pointer definition control block pointed to by the PtrDefRec parameter. The word 0 (buffer length = cb) pointer definition record parameter field must contain the size in bytes of the application buffer where the device driver is to insert the sessions pointer image. All other words in the parameter list are returned to the application by MouGetPtrShape.

If the buffer size is insufficient, the TotLength field contains the actual size in bytes of the returned pointer image.

The pointer shape may be set by the application with MouSetPtrShape or may be the default image provided by the installed Pointer Device Driver.

# MouGetScaleFact (xPM)

## Description

This call returns a pair of 1-word scaling factors for the current mouse device.

```
#define INCL_MOU
#include <os2.h>

USHORT MouGetScaleFact (PSCALEFACT pmouscFactors, HMOU hmou);
```

`pmouscFactors (PSCALEFACT)` - output
: Address of the control block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K – 1).

  `rowScale (USHORT)`
  : Row scaling factor.

  `colScale (USHORT)`
  : Column scaling factor. See `MouSetScaleFact` for more information.

`hmou (HMOU)` - input
: Contains the handle of the mouse device obtained from a previous `MouOpen`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

The units of the scale factor depend on the mode of the display screen for the session. If the screen is operating in text mode, the scaling units are relative to characters. If the screen is operating in graphics mode, the scaling units are relative to pels.

# MouInitReal (xWPM)

## Description

This call initializes mouse pointer draw support for DOS mode.

```
#define INCL_MOU
#include <os2.h>

USHORT MouInitReal (PSZ driverName);
```

`driverName (PSZ)` - input

Address of the name of the Pointer Draw Device Driver used as the pointer-image drawing routine for the DOS mode session.

The name of the device driver must be included in the CONFIG.SYS file at system start-up time.

If the selector portion of the far address is zero and the offset portion is non-zero, the offset portion identifies the power-up display configuration.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 412 | ERROR_MOUSE_SMG_ONLY |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouInitReal` is issued by the Base Video Subsystem at system initialization time.

The DOS mode mouse API (INT 33H), in contrast to the OS/2 mode Mouse API, does not contain an OPEN command. In addition, there is only one session for DOS mode.

The default pointer draw routine for DOS mode is located in the same pointer draw device driver, POINTER$, that is used for OS/2 mode. Establishing addressability to the pointer draw routine must be done during system initialization. This requires passing the entry point of the DOS mode pointer draw routine to the mouse device driver. This is the purpose of the `MouInitReal` call. It passes the address of the default, power-up

pointer draw routine for DOS mode to the mouse device driver. This initialization is transparent to applications.

This call is for use only by the Base Video Subsystem when invoked during system initialization under the shell/session manager PID.

The error code ERROR_MOUSE_SMG_ONLY is valid from shell process only.

# MouOpen (xPM)

## Description

This call opens the mouse device for the current session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouOpen (PSZ pszDvrName, PHMOU phmou);
```

`pszDvrName (PSZ)` - input

`DriverName` is a far pointer to an ASCIIZ string in application storage containing the name of the pointer draw device driver to be used as the pointer-image drawing routine for this session.

The name of the device driver must be included in the CONFIG.SYS file at system start-up time. Applications that use the default pointer draw device driver supplied by the system must push a double-word of 0s in place of an address.

`DriverName` has a different definition when the caller is the Base Video Subsystem (BVS). In this case the selector portion of the far address is zero. The offset portion is non-zero and contains a display configuration number (sequentially numbered where 1 is the first display configuration). The `MouOpen` call issued by BVS is executed on the `VioSetMode` path. Using the display configuration number passed on the `MouOpen` call, the Base Mouse Subsystem can detect a change in display configurations. This form of the `MouOpen` call is not recommended for applications. Applications should either push the far address of an ASCIIZ pointer draw device driver name or push two words of zeros.

`phmou (PHMOU)` - output

Address of a 1-word value that represents the mouse handle returned to the application.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 390 | ERROR_MOUSE_INV_MODULE_PT |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouOpen` initializes the Mouse functions to a known state. The application may have to issue additional mouse functions to establish the environment it desires. For example, after the `MouOpen`, the collision area is defined to be the size of the entire display. Therefore, to get the pointer to be displayed, the application must issue a `MouDrawPtr` to remove the collision area.

The state of the mouse after the first `MouOpen` is:

- Row/Col scale factors set to 16/8. (See `MouSetScaleFact`.)
- All events reported. (See `MouSetEventMask`.)
- Empty event queue. (See `MouReadEventQue` and `MouGetNumQueEl`.)
- All user settable Device Status bits reset. (Set to zero. See `MouSetDevStatus`.)
- Pointer set to center of screen if valid display mode is set. (See `MouSetPtrPos`.)
- Pointer shape set to the default for the pointer device driver currently registered in the session. (See `MouSetPtrShape`.)
- Collision area equal to full screen. (See `MouDrawPtr` and `MouRemovePtr`.)

# MouReadEventQue (xPM)

## Description

This call reads an event from the mouse device FIFO event queue, and places it in a structure provided by the application.

```
#define INCL_MOU
#include <os2.h>

USHORT MouReadEventQue (PMOUEVENTINFO pmouevEvent, PUSHORT pfWait,
                        HMOU hmou);
```

`pmouevEvent (PMOUEVENTINFO)` - output

Address of the status of the mouse event queue.

`fs (USHORT)`

State of the mouse at the time of the event.

| Bit | Description |
|---|---|
| 15–7 | Reserved, set to zero. |
| 6 | Set to report button 3 press/release events, without mouse motion.<br>(fs & MOUSE_BN3_DOWN) |
| 5 | Set to report button 3 press/release events, with mouse motion.<br>(fs & MOUSE_MOTION_WITH_BN3_DOWN) |
| 4 | Set to report button 2 press/release events, without mouse motion.<br>(fs & MOUSE_BN2_DOWN) |
| 3 | Set to report button 2 press/release events, with mouse motion.<br>(fs & MOUSE_MOTION_WITH_BN2_DOWN) |
| 2 | Set to report button 1 press/release events, without mouse motion.<br>(fs & MOUSE_BN1_DOWN) |
| 1 | Set to report button 1 press/release events, with mouse motion.<br>(fs & MOUSE_MOTION_WITH_BN1_DOWN) |
| 0 | Set to report mouse motion events with no button press/release events.<br>(fs & MOUSE_MOTION) |

`time (ULONG)`

Time stamp (in milliseconds) since the system was started.

`row (USHORT)`

Absolute or relative row position.

`col (USHORT)`
Absolute or relative column position.

`pfWait (PUSHORT)` - input
Address of the action to take when `MouReadEventQue` is issued and the mouse event queue is empty. If the mouse event queue is not empty, this parameter is not examined by the mouse support. `Read-Type` values are:

| Value | Definition |
|---|---|
| 0 | No Wait for data on empty queue (return a NULL record) (*pfWait == MOU_WAIT) |
| 1 | WAIT for data on empty queue. (*pfWait == MOU_NOWAIT) |

`hmou (HMOU)` - input
Handle of the mouse device from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 387 | ERROR_MOUSE_INV_PARMS |
| 393 | ERROR_MOUSE_NO_DATA |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

The types of queued events are directly affected by the current value of the Mouse `EventMask`. `MouSetEventMask` is used to indicate the types of events desired, and `MouGetEventMask` is used to query the current value of the mask. Refer to these functions for further explanation of the masking of events.

Recognition of the mouse transition depends on the use of `MouState` returned in the event record. The application should focus on bit transitions that occur in this word. It is important to properly set the event mask with `MouSetEventMask` for reporting the state transitions.

`MouState` reports the state of the mouse that resulted from the action that caused the event. The action can be pressing or releasing a button, and/or moving the mouse. All status is given, regardless of the `EventMask` that was used to determine whether or not to report the event.

For example, assume the `EventMask` indicates that the application wants only button 1 event. The `EventMask` has only bits 1 and 2 set in this case. Also assume the current state of the mouse is no buttons down, and mouse is not moving. At this point, button 1 is pressed causing an event; the status shows button 1 down (bit 2 set). Next the mouse is moved,

thereby causing more events; status shows bit 1 set. Finally, mouse is stopped and button 1 is released. The event shows status with no bits set. Next, button 2 is pressed. No event occurs. Mouse is then moved; again, no event. Then, while mouse is still in motion, button 1 is pressed; an event is generated with bits 1 and 3 set in the state word. While mouse is still in motion, both buttons are released. Because button 1 changes states, an event occurs. The state word has bit 0 set. Finally, mouse is stopped. No event occurs, again because no button 1 transition has taken place.

**Note** We have found that the mouse queue does not, in fact, behave as described above by IBM. Our tests have been in OS/2 2.0 + Service Pack. These quirks might be corrected in a future revision. Specifically, we have found the following differences:

- If events for button 3 are enabled but movement events are disabled and the mouse has only 2 buttons, movement events will still be queued.
- If, for example, button 1 is enabled, and button 2 is disabled, the button 2 state will be reported as up on a button 1 press irrespective of whether it is actually up or down. If a button 1 down-and-moving event occurs, thereafter, for the duration of the button 1 press, button 2 events will be reported.

As such, we recommend that mouse button events not be masked out using the `EventMask`. Rather, all events for all existing buttons should be accepted and, if necessary, filtered by the application.

The `Row` and `Column` fields in the `Buffer` parameter may contain either absolute display coordinates or relative mouse motion in mickeys. See `MouSetDevStatus` for additional information.

# MouRegister (xWPM)

## Description

This call registers a mouse subsystem within a session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouRegister (PSZ pszModName, PSZ pszEntryName, ULONG flFuns);
```

`pszModName (PSZ)` - input
: Address of the dynamic link module name. The maximum length is 9 bytes (including ASCIIZ terminator).

`pszEntryName (PSZ)` - input
: Address of the dynamic link entry point name of a routine that receives control when any of the registered functions are called. The maximum length is 33 bytes (including ASCIIZ terminator).

`flFuns (ULONG)` - input
: A mask of bits, where each bit set to 1 identifies a mouse function being registered. Bit values are

| Bit | Registered function | Mask |
|---|---|---|
| 31–22 | Reserved, set to zero | |
| 20 | MouSetDevStatus | (flFuns & MR_MOUSETDEVSTATUS) |
| 19 | MouInitReal | (flFuns & MR_MOUINITREAL) |
| 18 | MouSetPtrPos | (flFuns & MR_MOUSETPTRPOS) |
| 17 | MouGetPtrPos | (flFuns & MR_MOUGETPTRPOS) |
| 16 | MouRemovePtr | (flFuns & MR_MOUREMOVEPTR) |
| 15 | MouDrawPtr | (flFuns & MR_MOUDRAWPTR) |
| 14 | MouSetPtrShape | (flFuns & MR_MOUSETPTRSHAPE) |
| 13 | MouGetPtrShape | (flFuns & MR_MOUGETPTRSHAPE) |
| 12 | MouClose | (flFuns & MR_MOUCLOSE) |
| 11 | MouOpen | (flFuns & MR_MOUOPEN) |
| 10 | Reserved | |
| 9 | Reserved | |
| 8 | MouSetEventMask | (flFuns & MR_MOUSETEVENTMASK) |
| 7 | MouSetScaleFact | (flFuns & MR_MOUSETSCALEFACT) |
| 6 | MouGetEventMask | (flFuns & MR_MOUGETEVENTMASK) |
| 5 | MouGetScaleFact | (flFuns & MR_MOUGETSCALEFACT) |
| 4 | MouReadEventQue | (flFuns & MR_MOUREADEVENTQUE) |
| 3 | MouGetNumQueEl | (flFuns & MR_MOUGETNUMQUEEL) |
| 2 | MouGetDevStatus | (flFuns & MR_MOUGETDEVSTATUS) |
| 1 | MouGetNumMickeys | (flFuns & MR_MOUGETNUMMICKEYS) |
| 0 | MouGetNumButtons | (flFuns & MR_MOUGETNUMBUTTONS) |

## Return value

```
  0  NO_ERROR
385  ERROR_MOUSE_NO_DEVICE
413  ERROR_MOUSE_INVALID_ASCIIZ
414  ERROR_MOUSE_INVALID_MASK
415  ERROR_MOUSE_REGISTER
466  ERROR_MOU_DETACHED
505  ERROR_MOU_EXTENDED_SG
```

## Remarks

The Base Mouse Subsystem is the default mouse subsystem. There can be only one `MouRegister` outstanding for each session without an intervening `MouDeRegister`. `MouDeRegister` must be issued by the same process that issued `MouRegister`.

When any registered function is called, control is routed to `EntryName`. When this routine is entered, four additional values are pushed onto the stack. The first is the index number (Word) of the function being called. The second is a near pointer (Word). The third is the caller's DS register (Word). The fourth is the return address (DWord) to the mouse router. For example, if `MouGetNumMickeys` were called and control routed to `EntryName`, the stack would appear as if the following instructions were executed:

```
PUSH@ WORD  NumberOfMickeys
PUSH  WORD  DeviceHandle
CALL  FAR   MouGetNumMickeys
PUSH  WORD  Function Code
CALL  NEAR  Entry point in Mouse Router
PUSH  DS
CALL  FAR   EntryName
```

When a registered function returns to the Mouse Router, AX is interpreted as follows:

**AX = 0**

No error. Do not invoke the Base Mouse Subsystem routine. Return AX = 0.

**AX = -1**

Invoke the BaseMouse Subsystem routine. Return AX = return code from the Base Mouse Subsystem.

**AX = error (if not 0 or -1)**

Do not invoke the Base Mouse Subsystem Routine. Return AX = error.

When the mouse router receives a mouse call, it routes it to the Base Mouse Subsystem unless an application or other mouse subsystem has previously issued `MouRegister` for that call. If the call was registered, the subsystem is entered at the `EntryName` specified, and provided with the applicable function code.

The registered function mask is used to determine whether a requested function is performed by the registered mouse subsystem or default to the Base Mouse Subsystem.

The following list shows the relationship of the mouse API calls and the function code passed to either the Base Mouse Subsystem or a registered mouse subsystem.

| MOU API calls | Function code |
|---|---|
| `MouGetNumButtons` | 00H |
| `MouGetNumMickeys` | 01H |
| `MouGetDevStatus` | 02H |
| `MouGetNumQueEl` | 03H |
| `MouReadEventQue` | 03H |
| `MouGetScaleFact` | 05H |
| `MouGetEventMask` | 06H |
| `MouSetScaleFact` | 07H |
| `MouSetEventMask` | 08H |
| Reserved | 09H |
| Reserved | 0AH |
| `MouOpen` | 0BH |
| `MouClose` | 0CH |
| `MouGetPtrShape` | 0DH |
| `MouSetPtrShape` | 0EH |
| `MouDrawPtr` | 0FH |
| `MouRemovePtr` | 10H |
| `MouGetPtrPos` | 11H |
| `MouSetPtrPos` | 12H |
| `MouInitReal` | 13H |
| `MouFlushQue` | 14H |
| `MouSetDevStatus` | 15H |

A registered mouse subsystem must leave the stack, on exit, in the exact state it was received.

# MouRemovePtr (xPM)

## Description

This call allows a process to notify the mouse device driver that the area defined by the passed parameters is for the exclusive use of the application. This area is defined as the collision area and is not available to the mouse device driver when drawing pointer images.

```
#define INCL_MOU
#include <os2.h>

USHORT MouRemovePtr (PNOPTRRECT pmourtRect, HMOU hmou);
```

`pmourtRect (PNOPTRRECT)` - input
: Address of the pointer shape collision area structure:

  `row (USHORT)`
  : Upper left row coordinate (pels or characters).

  `col (USHORT)`
  : Upper left column coordinate (pels or characters).

  `cRow (USHORT)`
  : Lower right row coordinate (pels or characters).

  `cCol (USHORT)`
  : Lower right column coordinate (pels or characters).

`hmou (HMOU)` - input
: Handle of the mouse device from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 387 | ERROR_MOUSE_INV_PARMS |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouRemovePtr` may be issued by any process in the session. However, only one collision area is active at a time. Each `MouRemovePtr` command has the effect of resetting the collision area to the location and area specified by the current command.

If the logical pointer position is outside of the collision area specified by the latest `MouRemovePtr` command, the pointer image is drawn.

The `MouDrawPtr` command effectively cancels the `MouRemovePtr` command and allows the pointer to be drawn anywhere on the screen, until a new `MouRemovePtr` command is issued.

# MouSetDevStatus (xPM)

## Description

This call sets the mouse device driver status flags for the installed mouse device driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouSetDevStatus (PUSHORT pfsDevStatus, HMOU hmou);
```

`pfsDevStatus (PUSHORT)` - input
Address of the desired status flag settings.

The passed parameter is a 2-byte set of flags. Only the high-order byte has meaning.

| Bit | Description |
|---|---|
| 15–10 | Reserved, set to zero. |
| 9 | Set if mouse device is to return data in mickeys. (*pfsDevStatus & MOUSE_MICKEYS) |
| 8 | Set if the drawing operations for the pointer draw routine are to be disabled. (*pfsDevStatus & MOUSE_DISABLED) |
| 7–0 | Reserved, set to zero. |

`hmou (HMOU)` - input
Handle of the mouse device from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 387 | ERROR_MOUSE_INV_PARMS |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouSetDevStatus` is the complement to `MouGetDevStatus`. However, not all status flags may be set with `MouSetDevStatus`. Only the flags corresponding to the following functions may be modified:

- Return data in mickeys. Normally, mouse data is returned to the application with the absolute display mode coordinates of the pointer im-

age position on the display screen. By setting this status flag, mouse data is returned in relative mickeys, a unit of mouse movement.

- Don't call pointer draw device. Normally, the pointer draw device driver is called for all drawing operations. By setting this status flag, the mouse device driver does not call the pointer draw device driver. The application must draw any required pointer image on the screen.

# MouSetEventMask (xPM)

## Description

This call assigns a new event mask to the current mouse device driver.

```
#define INCL_MOU
#include <os2.h>

USHORT MouSetEventMask (PUSHORT pfsEvents, HMOU hmou);
```

`pfsEvents (PUSHORT)` - input

Address of a value in application storage used to indicate what mouse events are to be placed on the event queue (see `MouReadEventQue`) and which events are to be ignored.

The `EventMask` bit values are described here:

| Bit | Description |
|---|---|
| 15–7 | Reserved, set to zero. |
| 6 | Set to report button 3 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN3_DOWN) |
| 5 | Set to report button 3 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN3_DOWN) |
| 4 | Set to report button 2 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN2_DOWN) |
| 3 | Set to report button 2 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN2_DOWN) |
| 2 | Set to report button 1 press/release events, without mouse motion.<br>(*pfsEvents & MOUSE_BN1_DOWN) |
| 1 | Set to report button 1 press/release events, with mouse motion.<br>(*pfsEvents & MOUSE_MOTION_WITH_BN1_DOWN) |
| 0 | Set to report mouse motion events with no button press/release events.<br>(*pfsEvents & MOUSE_MOTION) |

A bit clear setting (set to zero) in an `EventMask` bit position indicates that the associated type of event is not reported to the application. Note also that the mouse buttons are numbered from left to right for a right-handed mouse and right to left for a left-handed mouse. When the mouse is properly positioned for use, the forefinger button is button 1.

`hmou (HMOU)` - input
Handle of the mouse device from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

Setting a bit in the event mask means that the associated event is reported on the mouse FIFO event queue. See `MouReadEventQue` for examples of event mask use.

# MouSetPtrPos (xPM)

## Description

This call directs the mouse driver to set a new row and column coordinate position for the mouse pointer.

```
#define INCL_MOU
#include <os2.h>

USHORT MouSetPtrPos (PPTRLOC pmouLoc, HMOU hmou);
```

`pmouLoc (PPTRLOC)` - input
Address of the mouse pointer position structure:

`row (USHORT)`
New pointer row coordinate (pels or characters).

`col (USHORT)`
New pointer column coordinate (pels or characters).

`hmou (HMOU)` - input
Handle of the mouse device from a previous MouOpen.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 387 | ERROR_MOUSE_INV_PARMS |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

The application must ensure that the coordinate position specified conforms to the current display mode orientation for the session. Pel values must be used for graphics modes and character values for text modes.

This function has no effect on the display's current collision area definition as specified by the `MouDrawPtr` call. If the mouse pointer image is directed into a defined collision area, the pointer image is not drawn until either the pointer is moved outside the collision area or the collision area is released by the `MouDrawPtr` call.

## Description

This call allows a process to set the pointer shape and size to be used as the mouse device driver pointer image for all applications in a session.

```
#define INCL_MOU
#include <os2.h>

USHORT MouSetPtrShape (PBYTE pBuf, PPTRSHAPE pmoupsInfo, HMOU hmou);
```

`pBuf (PBYTE)` - input

Address of a buffer containing the bit image used by the mouse device driver as the pointer shape for that session. The buffer consists of AND and XOR pointer masks in a format meaningful to the pointer draw device driver.

For CGA compatible text modes (0, 1, 2, and 3) the following describes the AND and XOR pointer mask bit definitions for each character cell of the masks. Bit values are

| Bit | Description |
|---|---|
| 15 | Blinking |
| 14–12 | Background color |
| 11 | Intensity |
| 10–8 | Foreground color |
| 7–0 | Character |

`pmoupsInfo (PPTRSHAPE)` - input

Address of the structure where the application stores the necessary data for the pointer draw device driver to build a row-by-column image for each bit plane for the current display mode. The pointer definition record structure follows:

`cb (USHORT)`

Length of the pointer buffer available for the pointer device driver to build a Row by Col image for each bit plane for the mode the display is currently running. This value is supplied by the application. If the value is too small, pointer draw places the true length of the image in this field, and returns an error.

For all OS/2 system-supported modes, `TotLength` is specified in bytes and is equal to

**Mono & Text Modes**

For text mode height and width must be 1, so length is always 4.

cb = (height in chars) * (width in chars) * 2 * 2
= 1 * 1 * 2 * 2
= 4

**Graphics mode**

Width-in-pels must be a multiple of 8.

cb = (height in pels)*(width in pels)*(bits per pel)*2/8

**Modes 4 and 5 (320 × 200)**

cb = (height) * (width) * 2 * 2 / 8

**Mode 6 (640 × 200)**

cb = (height) * (width) * 1 * 2 / 8

Length calculations produce byte boundary buffer sizes.

Number of columns in the mouse shape. In graphics modes, this field contains the pel width (columns) of the mouse shape for the session and must be greater than or equal to 1. In text modes, col must equal 1.

`row (USHORT)`

Number of rows in the mouse shape. In graphics modes, this field contains the pel height (rows) of the mouse shape for the session and must be greater than or equal to 1. In text modes, row must equal 1.

`colHot (USHORT)`

This value is returned by the mouse device driver to indicate the relative column offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

`rowHot (USHORT)`

This value is returned by the mouse device driver to indicate the relative row offset within the pointer image. The value defines the center (hotspot) of the pointer image. This value is a signed number that represents either character or pel offset, depending on the display mode.

**Note** For other custom displays and for the extended modes of the EGA attachment, it is possible to set the display to modes that require multiple bit planes. In these cases, the area sized by the row and column limits must be repeated for each bit plane supported in that mode. Consequently, the calling process must supply enough data to allow the mouse device driver to draw the pointer shape on all currently supported bit planes in that session. For text modes, row and column offset must equal 0.

`hmou (HMOU)` - input
Contains the handle of the mouse device obtained from a previous `MouOpen`.

## Return value

0 NO_ERROR
385 ERROR_MOUSE_NO_DEVICE
387 ERROR_MOUSE_INV_PARMS
466 ERROR_MOU_DETACHED
501 ERROR_MOUSE_NO_CONSOLE
505 ERROR_MOU_EXTENDED_SG

## Remarks

An application passes a data image to the mouse device driver that the mouse driver applies to the screen whenever the logical pointer position is not located in the application-defined collision area. The application synchronizes use of the screen with the mouse driver by way of `MouRemovePtr` and `MouDrawPtr`.

The pointer shape is dependent on the display device driver used to support the display device. OS/2 supports text and graphics modes. These modes are restricted to modes 0 through 7, depending on the display device. Character modes (modes 0, 1, 2, 3, and 7) support the pointer cursor only as a reverse block character. This reverse block character has a character height and width equal to 1.

The pointer shape is mapped by the Pointer Draw Device Driver and determined completely by the application. The height and width may vary from 1 through the pel size of the display screen. For restrictions concerning the Pointer Draw Device Driver, see *IBM Operating System/2.0 Technical Library Physical Device Driver Reference*.

# MouSetScaleFact (xPM)

## Description

This call assigns to the current mouse device driver a new pair of 1-word scaling factors.

```
USHORT MouSetScaleFact (PSCALEFACT pmouscFactors, HMOU hmou);
```

`pmouscFactors (PSCALEFACT)` - **input**
Address of the control block structure that contains the current row and column coordinate scaling factors. The scaling factors must be greater than or equal to 1 and less than or equal to (32K – 1).

`rowScale (USHORT)`
Row scaling factor.

`colScale (USHORT)`
Column scaling factor.

`hmou (HMOU)` - **input**
Handle of the mouse device from a previous `MouOpen`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 385 | ERROR_MOUSE_NO_DEVICE |
| 387 | ERROR_MOUSE_INV_PARMS |
| 466 | ERROR_MOU_DETACHED |
| 501 | ERROR_MOUSE_NO_CONSOLE |
| 505 | ERROR_MOU_EXTENDED_SG |

## Remarks

`MouSetScaleFact` sets the mickey-to-pixel ratio for mouse motion. The row scale and column scale ratios specify a number of mickeys for each 8 pixels. The default value for the row scale is 16 mickeys for each 8 pixels. The default value for the column scale is 8 mickeys to 8 pixels.

The number of pixels moved does not have to correspond 1-to-1 with the number of mickeys the mouse moves. The scaling factor defines a sensitivity for the mouse that is a ratio of the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines at what rate the cursor moves on the screen.

# MouSynch (xWPM)

## Description

This call provides synchronous access for a mouse subsystem to the mouse device driver.

```
USHORT MouSynch(USHORT ioWait);
```

`ioWait (USHORT)` - **input**
Wait for access. The flag Word is defined as follows:

| Value | Definition |
|---|---|
| 0 | Control immediately returned to requestor. |
| 1 | Requestor waits until mouse device driver is free. |

## Return value

0 NO_ERROR
121 ERROR_SEM_TIMEOUT

## Remarks

`MouSynch` blocks all other threads within a session until the semaphore clears (returns from the subsystem to the router). To ensure proper synchronization, `MouSynch` should be issued by a mouse subsystem if it intends to access dynamically modifiable shared data for each session or if it intends to issue a `DosDevIOCtl`. `MouSynch` does not protect globally shared data from threads in other sessions.

# Video (VIO) functions

This part describes the OS/2 Video API interface.

If ERROR_VIO_SEE_ERROR_LOG is returned, further information about the error that occurred can be obtained by calling `WinGetLastError`.

## Notes

- Calls marked xPM are not supported by Presentation Manager, and must not be used by Presentation Manager applications. An error code is returned if any of these calls are issued.
- Calls marked xWPM are not windowable and are not supported by Presentation Manager. They can be used in OS/2 mode.
- Calls marked FAPI are present in the Family API.

**Table 3-1 Video call list**

| Function call | Icon |
|---|---|
| `VioDeRegister` | xWPM |
| `VioGetConfig` | FAPI |
| `VioGetCurPos` | FAPI |
| `VioGetCurType` | FAPI |
| `VioGetFont` | FAPI xWPM |
| `VioGetMode` | FAPI xPM |
| `VioGetPhysBuf` | FAPI xWPM |
| `VioGetState` | FAPI xWPM |
| `VioModeUndo` | xWPM |
| `VioModeWait` | xWPM |
| `VioPrtSc` | xWPM |
| `VioPrtScToggle` | xWPM |
| `VioReadCellStr` | FAPI |
| `VioReadCharStr` | FAPI |
| `VioRegister` | xWPM |
| `VioSavRedrawUndo` | xWPM |
| `VioSavRedrawWait` | xWPM |
| `VioScrLock` | FAPI xWPM |
| `VioScrollDn` | FAPI |
| `VioScrollLf` | FAPI |
| `VioScrollRt` | FAPI |
| `VioScrollUp` | FAPI |
| `VioScrUnLock` | FAPI xWPM |
| `VioSetCurPos` | FAPI |
| `VioSetCurType` | FAPI |
| `VioSetFont` | FAPI xWPM |
| `VioSetMode` | FAPI xPM |
| `VioSetState` | FAPI xWPM |

| | |
|---|---|
| VioWrtCellStr | FAPI |
| VioWrtCharStr | FAPI |
| VioWrtCharStrAtt | FAPI |
| VioWrtNAttr | FAPI |
| VioWrtNCell | FAPI |
| VioWrtNChar | FAPI |
| VioWrtTTY | FAPI |

# VioDeRegister (xWPM)

## Description

This call deregisters a video subsystem previously registered within a session.

```
#define INCL_VIO
#include <os2.h>

USHORT VioDeRegister (void);
```

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 404 | ERROR_VIO_DEREGISTER |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

`VioDeRegister` must be issued by the same process that issued the previous `VioRegister`. After `VioDeRegister` is issued, subsequent video calls are processed by the Base Video Subsystem.

# VioEndPopUp (xPM)

## Description

This call is issued by the application when it no longer requires the temporary screen obtained through a previous `VioPopUp` call.

```
#define INCL_VIO
#include <os2.h>

USHORT VioEndPopUp (HVIO hvio);
```

`hvio (HVIO)` - input
A reserved word of 0s.

## Return value

0 NO_ERROR
405 ERROR_VIO_NO_POPUP
436 ERROR_VIO_INVALID_HANDLE

## Remarks

When the application issues a `VioEndPopUp` call, all video calls are directed to the application's normal video buffer.

## PM considerations

An error is returned if issued with a non-zero handle.

# VioGetAnsi (xPM)

## Description

This call returns the current ANSI status On/Off state.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetAnsi (PUSHORT pfAnsi, HVIO hvio);
```

`pfAnsi (PUSHORT)` - output
Address of the current ANSI status. A value of 1 indicates ANSI is active, and a value of 0 indicates ANSI is not active.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Description

This call returns the address of the logical video buffer (LVB).

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetBuf (PULONG pLVB, PUSHORT pcbLVB, HVIO hvio);
```

`pLVB (PULONG)` - output
: Address of the selector and offset of the logical video buffer. Applications should not assume the offset portion of this far address is 0.

`pcbLVB (PUSHORT)` - output
: Address of the length buffer in bytes. The length is the number of rows multiplied by number of columns multiplied by size of cell.

`hvio (HVIO)` - input
: This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

An application using `VioGetBuf` can prepare a screen in the application's own logical video buffer (LVB) offline. When the application is in the foreground, the physical screen buffer is updated from LVB when `VioShowBuf` is issued. When the application runs in the background, the physical screen buffer is updated when the application is switched to the foreground.

Once `VioGetBuf` is issued, all VioWrtXX calls issued while the application is running in the foreground are written to the physical display buffer and LVB. If a `VioGetPhysBuf` is subsequently issued, then the `VioWrtXX` calls are only written to the physical display buffer. They are no longer written to the LVB.

`VioGetMode` may be used to determine the dimensions of the buffer.

If `VioSetMode` is issued following a `VioGetBuf` call, the size of the logical video buffer is adjusted to correspond to the new mode. There is one logical video buffer per session (or presentation space if AVIO application) that corresponds to the current mode on the current display configuration.

## PM considerations

This function returns the address and length of the Advanced VIO presentation space. The presentation space may be used to directly manipulate displayed information.

## Description

This call returns the video display configuration.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetConfig (USHORT usConfigId, PVIOCONFIGINFO pvioin,
HVIO hvio);
```

`usConfigId (USHORT)` - input

Identifies for which display configuration information is being requested:

| Value | Definition |
|---|---|
| 0 | Current configuration (usConfigId == VIO_CONFIG_CURRENT) |
| 1 | Primary configuration (usConfigId == VIO_CONFIG_PRIMARY) |
| 2 | Secondary configuration. (usConfigId == VIO_CONFIG_SECONDARY) |

`pvioin (PVIOCONFIGINFO)` - output

Address of structure where the display configuration is returned.

`cb (USHORT)`

The maximum size structure required is variable and can be determined by issuing `VioGetConfig` with Length set to 2. When cb is set to 2 on input, `VioGetConfig` returns the size of the maximum structure required in the cb field on output. When cb is not equal to 2 on input, the cb field is modified on output to reflect the actual number of bytes returned. That is, if more than the maximum size was specified, the maximum size is returned. However, if less than the maximum size is specified, the value returned reflects the number of bytes of complete fields returned.

`adapter (USHORT)`

Display adapter type.

| Value | Definition |
|---|---|
| 0 | Monochrome-compatible (adapter == DISPLAY_MONOCHROME) |
| 1 | Color Graphics Adapter (CGA) (adapter == DISPLAY_CGA) |
| 2 | Enhanced Graphics Adapter (EGA) (adapter == DISPLAY_EGA) |

| Value | Definition |
|---|---|
| 3 | VGA or PS/2 Display Adapter<br>(adapter == DISPLAY_VGA) |
| 4–6 | Reserved |
| 7 | IBM Personal System/2 Display Adapter 8514/A.<br>(adapter == DISPLAY_8514A) |
| 8 | IBM PS/2 Image Adapter/A<br>(adapter == DISPLAY_IMAGEADAPTER) |
| 9 | IBM PS/2 XGA Display Adapter<br>(adapter == DISPLAY_XGA) |

Values ranging from 0–4095 are reserved for IBM.

`display (USHORT)`
Display or monitor type.

| Value | Definition |
|---|---|
| 0 | Monochrome display<br>(display == MONITOR_MONOCHROME) |
| 1 | Color display<br>(display == MONITOR_COLOR) |
| 2 | Enhanced Color Display<br>(display == MONITOR_ENHANCED) |
| 3 | PS/2 Monochrome Display 8503<br>(display == MONITOR_8503) |
| 4 | PS/2 Color Displays 8512 and 8513<br>(display == MONITOR_851X_COLOR) |
| 5–8 | Reserved |
| 9 | PS/2 Color Display 8514<br>(display == MONITOR_8514) |
| 10 | IBM Plasma Display Panel<br>(display == MONITOR_FLATPANEL) |
| 11 | Monochrome Displays 8507 and 8604<br>(display == MONITOR_8507_8604) |
| 12 | PS/2 Color Display 8515<br>(display == MONITOR_8515) |
| 13 | Reserved |

Values ranging from 0–4095 are reserved for IBM.

`cbMemory (ULONG)`
Amount of memory, in bytes, on the adapter.

`Configuration (USHORT)`
Number of the display configuration that this data corresponds to. This is assigned by the video subsystem, not the Base Video Handler (BVH).

`VDHVersion (USHORT)`
This field is reserved.

`Flags (USHORT)`
Bit field defined as follows:

| Bit | Description |
|---|---|
| 15–1 | Reserved |
| 0 | Power up display configuration. |

`HWBufferSize (ULONG)`
Size of the buffer required by the Base Video Handler (BVH) to save the full hardware state excluding the physical display buffer.

`FullSaveSize (ULONG)`
Maximum size buffer required by the BVH to save the full physical display buffer.

`PartSaveSize (ULONG)`
Maximum size buffer required by the BVH to save the portion of the physical display buffer that is overlaid by a pop-up.

`EMAdaptersOFF (USHORT)`
Offset within the configuration data structure to the following information describing what other display adapters are emulated by this display adapter.

`Number of Data words (USHORT)`
Contains a one-word field specifying a count of data words to follow.

`Data word 1 (USHORT)`
Bits set in the data words identify display adapters emulated. Data word 1 has the following definition:

| Bit | Description |
|---|---|
| 0 | Monochrome/printer adapter<br>(data[0] & (1 << DISPLAY_MONOCHROME)) |
| 1 | Color graphics adapter<br>(data[0] & (1 << DISPLAY_CGA)) |
| 2 | Enhanced graphics adapter<br>(data[0] & (1 << DISPLAY_EGA)) |
| 3 | VGA or PS/2 display adapter<br>(data[0] & (1 << DISPLAY_VGA)) |
| 4–6 | Reserved |
| 7 | 8514/A Adapter<br>(data[0] & (1 << DISPLAY_8514A)) |
| 8 | IBM PS/2 Image Adapter/A<br>(data[0] & (1 << DISPLAY_IMAGEADAPTER)) |
| 9 | IBM PS/2 XGA Adapter<br>(data[0] & (1 << DISPLAY_XGA)) |
| 10–15 | Reserved. |

`Data word 2 (USHORT)`
Reserved.

`Data word N (USHORT)`
Reserved.

`EMDisplaysOFF (USHORT)`
Offset within the configuration data structure to the following information describing what other displays are emulated by this display.

`Number of Data words (USHORT)`
One-word field specifying a count of data words to follow.

`Data word 1 (USHORT)`
Bits set in the data words identify displays emulated. Data word 1 has the following definition:

| Bit | Description |
|---|---|
| 0 | 5151 monochrome display |
| 1 | 5153 color display |
| 2 | 5154 enhanced color display |
| 3 | 8503 monochrome display |
| 4 | 8512 or 8513 color display |
| 5–8 | Reserved |
| 9 | 8514 color display |
| 10 | IBM Plasma Display Panel |
| 11 | Monochrome Displays 8507 and 8604 |
| 12 | 8515 color display |
| 13–15 | Reserved. |

`Data word 2 (USHORT)`
Reserved

`Data word N (USHORT)`
Reserved.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
421 ERROR_VIO_INVALID_PARMS
436 ERROR_VIO_INVALID_HANDLE
438 ERROR_VIO_INVALID_LENGTH
465 ERROR_VIO_DETACHED

## Remarks

The values returned may not be correct if the adapter cannot be properly identified by the Base Video Handler (BVH) selected at system installation time. It can also be incorrect if the physical setup does not match that indicated by the presence of the adapter or by adapter switches. For example, it is impossible to detect the absence of a display on a CGA or the display attached to an EGA, despite the setup switches.

# VioGetCp (xPM)

## Description

This call allows a process to query the code page currently used to display text data.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetCp (USHORT usReserved, PUSHORT pIdCodePage, HVIO hvio);
```

`usReserved (USHORT)` - input
A reserved word of 0s.

`pIdCodePage (PUSHORT)` - output
Address of a word in the application's data area. The current video code page is returned in this word.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED
468 ERROR_VIO_USER_FONT

## Remarks

The display code page ID previously set by `VioSetCp`, or inherited from the requesting process, is returned to the caller.

The code page tag returned is the currently active code page. A value of 0000 indicates that the code page in use is the ROM code page provided by the hardware.

If ERROR_VIO_USER_FONT is returned, it indicates a user font that was previously loaded with `VioSetFont` is the active code page.

# VioGetCurPos (FAPI)

## Description

This call returns the coordinates of the cursor.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetCurPos (PUSHORT pusRow, PUSHORT pusColumn, HVIO hvio);
```

`pusRow (PUSHORT)` - output
Address of the current Row position of the cursor where 0 is the top row.

`pusColumn (PUSHORT)` - output
Address of the current column position of the cursor where 0 is the leftmost column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

# VioGetCurType (FAPI)

## Description

This call returns the cursor type.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetCurType (PVIOCURSORINFO pvioCursorInfo, HVIO hvio);
```

`pvioCursorInfo (PVIOCURSORINFO)` - output
Address of the cursor characteristics structure:

`yStart (USHORT)`
Horizontal scan line in the character cell that marks the top line of the cursor. If the character cell has n scan lines, 0 is the top scan line of the character cell and (n-1) is the bottom scan line.

`cEnd (USHORT)`
Horizontal scan line in the character cell that marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined in yStart.

`cx (USHORT)`
Width of the cursor. In text modes, `cursorwidth` is the number of columns. The maximum number supported by the OS/2 base video subsystem is 1. In graphics modes, `cursorwidth` is the number of pels.

`attr (USHORT)`
A value of –1 denotes a hidden cursor, all other values in text mode denote normal cursor and in graphics mode denote color attribute.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If `CursorStartLine` and `CursorEndLine` were originally specified as percentages on `VioSetCurType` (using negative values), the positive values into

which they were translated are returned. Refer to `VioSetCurType` for more information on how percentages can be used to set `CursorStartLine` and `CursorEndLine` independent of the number of scan lines per character cell.

## Family API considerations

In DOS mode, `VioGetCurType` returns only two values for attr: 0 = visible cursor, and –1 = hidden cursor.

# VioGetFont (FAPI, xWPM)

## Description

This call returns either the font table of the size specified or the font in use.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetFont (PVIOFONTINFO pviofi, HVIO hvio);
```

`pviofi (PVIOFONTINFO)` - input/output
Address of the font structure that returns current RAM font or specified ROM or code page font depending on the request type:

`cb (USHORT)`
Length of structure, including cb.

14 Only valid value.

`type (USHORT)`
Request type:

| Value | Definition |
|---|---|
| 0 | Get current RAM font for EGA, VGA, or IBM Personal System/2 Display Adapter.<br>(type == VGFI_GETCURFONT) |
| 1 | Get ROM font for CGA, EGA, VGA, or IBM Personal System/2 Display Adapter.<br>(type == VGFI_GETROMFONT) |

`cxCell (USHORT)`
Pel columns in character cell.

`cyCell (USHORT)`
Pel rows in character cell.

`pbData (PVOID)`
Address of the requested font table returned in a caller-supplied data area. If the storage area is accessed by way of an address of 0, a system-supplied segment containing the requested font table is returned.

`cbData (USHORT)`
Length, in bytes, of the caller-supplied data area where the font table is returned.

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 421 | ERROR_VIO_INVALID_PARMS |
| 438 | ERROR_VIO_INVALID_LENGTH |
| 465 | ERROR_VIO_DETACHED |
| 467 | ERROR_VIO_FONT |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

For reqtype = 1, return ROM font, the font size requested must be supported by the display adapter installed. The 8×8, 8×14, 9×14, 8×16, or 9×16 character font may be requested for the VGA or PS/2 Display Adapters. The 8×8, 8×14, or 9×14 font may be requested for the enhanced graphics adapter. The 8×8 font may be requested for the color graphics adapter.

**Note** Although graphics mode support is provided in `VioGetFont`, this support is not provided by the Base Video Handlers provided with OS/2.

For reqtype = 1, return ROM font, the far address returned is a ROM pointer only for those fonts where the font table for the full 256-character set is actually contained in ROM. Otherwise, the far address returned is a RAM pointer. Note that for 8×8 on the CGA, the font table for the full 256-character set is returned. For 9×14 or 9×16 the font table for the full 256-character set is also returned. Partial fonts are not returned. The 9×14 and 9×16 fonts are derived from variations of the 8×14 and 8×16 fonts, respectively, where the definitions of fonts for those characters that are different are replaced.

For `VioGetFont` specifying reqtype = 1, return ROM font, the font returned is derived from the fonts contained in the system, EGA, VGA, and PS/2 Display Adapter BIOS data areas as applicable. There is an exception for the EGA, VGA and PS/2 Display Adapter when `VioSetCp` or `VioSetFont` has been issued. In that case, the font of the size requested is returned from the active code page or the list of user fonts already set.

# VioGetMode (FAPI, xPM)

## Description

This call returns the mode of the display.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetMode (PVIOMODEINFO pvioModeInfo, HVIO hvio);
```

`pvioModeInfo (PVIOMODEINFO)` - input/output
Far address of a structure where mode characteristics are returned.

`cb (USHORT)`
Input parameter to `VioGetMode`. Length specifies the length of the data structure in bytes including Length itself. The value specified on input controls the amount of mode data returned. The minimum structure size required is 2 bytes, and the maximum structure size required is 34 bytes. A length of 2 returns the size of the maximum structure required for all the mode data. When length is not equal to 2, the length field is modified on output to reflect the actual number of bytes returned.

`fbType (UCHAR)`
Mode characteristics bit mask:

| Bit | Description |
|---|---|
| 7–4 | Reserved |
| 3 | 0 = VGA-compatible modes 0 thru 13H<br>1 = Native mode |
| 2 | 0 = Enable color burst<br>!(fbType & VGMT_DISABLEBURST)<br>1 = Disable color burst<br>(fbType & VGMT_DISABLEBURST) |
| 1 | 0 = Text mode<br>!(fbType & VGMT_GRAPHICS)<br>1 = Graphics mode<br>(fbType & VGMT_GRAPHICS) |
| 0 | 0 = Monochrome compatible mode<br>!(fbType & VGMT_OTHER)<br>1 = Other.<br>(fbType & VGMT_OTHER) |

`color (UCHAR)`
Number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color, for example:

| Value | Definition |
|---|---|
| 0 | Monochrome modes 7, 7+, and F. |
| 1 | 2 colors |
| 2 | 4 colors |
| 4 | 16 colors |
| 8 | 256 colors |

`col (USHORT)`
Number of text columns.

`row (USHORT)`
Number of text rows.

`hres (USHORT)`
Horizontal resolution, number of pel columns.

`vres (USHORT)`
Vertical resolution, number of pel rows.

`fmt_ID (UCHAR)`
Format of the attributes.

`attrib (UCHAR)`
Number of attributes in a character cell.

`buf_addr (ULONG)`
32-bit physical address of the physical display buffer for this mode.

`buf_length (ULONG)`
Length of the physical display buffer for this mode.

`full_length (ULONG)`
Size of the buffer required for a full save of the physical display buffer for this mode.

`partial_length (ULONG)`
Size of the buffer required for a partial (pop-up) save of the physical display buffer for this mode.

`ext_data_addr (PCH)`
Far address to an extended mode data structure or zero if none. The format of the extended mode data structure is determined by the device driver and is unknown to OS/2.

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

0 NO_ERROR
436 ERROR_VIO_INVALID_HANDLE

438 ERROR_VIO_INVALID_LENGTH
465 ERROR_VIO_DETACHED
494 ERROR_VIO_EXTENDED_SG

## Remarks

Refer to `VioSetMode` for examples.

# VioGetPhysBuf (FAPI, xWPM)

## Description

This call gets addressability to the physical display buffer.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetPhysBuf (PVIOPHYSBUF pvioPhysBuf, USHORT usReserved);
```

`pvioPhysBuf (PVIOPHYSBUF)` - input/output

Address of the data structure that contains the physical display buffer address and length on input and returns the selectors used to address the display buffer.

`pBuf (PBYTE)`

Address of the 32-bit start address (selector:offset) of the physical display buffer passed as input. If displaybuflen is 0, then displaybufaddr is the far address of the `PhysBuf` block described below.

`cb (ULONG)`

32-bit length of the physical display buffer. If `displaybuflen` is 0, then `displaybufaddr` is treated as the far address of the `PhysBuf` block described later and the Selector List is not present.

`asel (SEL)`

Selector list.

Returns the selectors (each of word-length) that address the physical display buffer. The first selector returned in the list, addresses the first 64K of the physical display buffer or `displaybuflen`, whichever is smaller. If `displaybuflen` is greater than 64K, the second selector addresses the second 64K.

The last selector returned in the list addresses the remainder of the display buffer. The application is responsible for ensuring enough space is reserved for the selector list to accommodate the specified buffer length.

Layout of the `PhysBuf` block pointed to by `pBuf`. The `PhysBuf` block is a variable length data structure. The first word is the `Length` of the `PhysBuf` block in bytes. The remaining words of the structure are the selectors that address the physical video buffer. If `Length` is specified as 2, the required length of the `PhysBuf` Block is returned in its place.

`length (USHORT)`

Length of `PhysBuf` structure in bytes

`selector (SEL)`
First selector

`selector (SEL)`
Next selector

`selector (SEL)`
... ...

`selector (SEL)`
Last selector

`Reserved (USHORT)` - input
Reserved word of 0s.

## Return value

0 NO_ERROR
350 ERROR_VIO_PTR
429 ERROR_VIO_IN_BG
430 ERROR_VIO_ILLEGAL_DURING_POPUP
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED
494 ERROR_VIO_EXTENDED_SG

## Remarks

If `displaybuflen` = 0, `VioGetPhysBuf` returns a selector that addresses the physical display buffer corresponding to the current mode. One selector is returned in Selector List. If a `VioGetPhysBuf` is issued after a `VioGetBuf`, then all `VioWrtXX` calls will no longer be written to the LVB. They will only be written to the physical display buffer. An application uses `VioGetPhysBuf` to get addressability to the physical display buffer. The selector returned by `VioGetPhysBuf` may be used only when an application program is executing in the foreground. When an application wants to access the physical display buffer, the application must call `VioScrLock`. `VioScrLock` either waits until the program is running in the foreground or returns a warning when the program is running in the background. For more information refer to `VioScrLock` and `VioScrUnLock`.

The buffer range specified for the physical screen buffer must fall between hex 'A0000' and 'BFFFF' inclusive. An application may issue `VioGetPhysBuf` only when it is running in the foreground. An application may issue `VioGetPhysBuf` more than once.

# VioGetState (FAPI, xWPM)

## Description

This call returns the current settings of the palette registers, overscan (border) color, blink/background intensity switch, color registers, underline location, or target `VioSetMode` display configuration.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGetState (PVOID pState, HVIO hvio);
```

`pState (PVOID)` - input/output
Address of one of six different state structures, depending on the request type. The request type is the second word of the packet.

| Type | Definition |
|---|---|
| 0 | Get palette registers |
| 1 | Get overscan (border) color |
| 2 | Get blink/background intensity switch |
| 3 | Get color registers |
| 4 | Reserved |
| 5 | Get the scan line for underlining |
| 6 | Get target `VioSetMode` display configuration. |
| 7 | Reserved. |

The six structures, depending on request type, are:

**VIOPALSTATE**
Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including length.
38 Maximum valid value.

`type (USHORT)` - input
Request type 0 for palette registers.

`iFirst (USHORT)` - input
First palette register in the palette register sequence; must be specified in the range 0 through 15. The palette registers are returned in sequential order. The number returned is based upon cb.

`acolor (USHORT[(cb-6)/2])` - output
Color value for each palette register. The maximum number of entries in the color value array is 16.

**VIOOVERSCAN**

Applies to CGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 1 for overscan (border) color.

`color (USHORT)`- input
Color value.

**VIOINTENSITY**

Applies to CGA, EGA, MCGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 2 for blink/background intensity switch.

`fs (USHORT)` - output
Switch set as:

| Value | Definition |
|---|---|
| 0 | Blinking foreground colors enabled. |
| 1 | High intensity background colors enabled. |

**VIOCOLORREG**

Applies to VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
12 Length in bytes.

`type (USHORT)` - input
Request type 3 for color registers.

`firstcolorreg (USHORT)` - input
First color register to get in the color register sequence; must be specified in the range 0 through 255. The color registers are returned in sequential order.

`numcolorregs (USHORT)` - input
Number of color registers to get; must be specified in the range 1 through 256.

`colorregaddr (PCH)` - input
Far address of a data area where the color registers are returned. The size of the data area must be three bytes times the number of color registers to get. The format of each entry returned is as follows:

Byte 1 Red value
Byte 2 Green value
Byte 3 Blue value

**VIOSETULINELOC**

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

cb (USHORT) - input
Length of structure, including cb.
6 Length in bytes.

type (USHORT) - input
Request type 5 to get the scan line for underlining. Underlining is enabled only when the foreground color is 1 or 9.

scanline (USHORT) - output
The value returned is in the range 0 through 31 and is the scan line minus 1. A value of 32 means underlining is disabled.

**VIOSETTARGET**

cb (USHORT) - input
Length of structure, including cb.
6 Length in bytes.

type (USHORT) - input
Request type 6 to get display configuration selected to be the target of the next VioSetMode.

defaultalgorithm (USHORT) - output
Configuration:

| Value | Definition |
|---|---|
| 0 | Default selection algorithm. See VioSetMode. |
| 1 | Primary |
| 2 | Secondary. |

hvio (HVIO) - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 421 | ERROR_VIO_INVALID_PARMS |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 438 | ERROR_VIO_INVALID_LENGTH |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Family API considerations

Request type = 6, Get Target VioSetMode Display Configuration, and request type = 5, Get Underline Location, are not supported in the Family API.

# VioGlobalReg (xWPM)

## Description

`VioGlobalReg` allows a subsystem to receive notification at the completion of VIO calls issued by all applications running in full-screen sessions.

```
#define INCL_VIO
#include <os2.h>

USHORT VioGlobalReg(PSZ pszModName, PSZ pszEntryName, ULONG flFun1,
                    ULONG flFun2,USHORT usReturn);
```

`pszModName (PSZ)` - input
: Address of the ASCIIZ string containing the 1–8 character filename of the subsystem. The maximum length of the ASCIIZ string is 9 bytes including the terminating byte of zero. The module must be a dynamic link library but the name supplied must not include the .DLL extension.

`pszEntryName (PSZ)` - input
: Address of the ASCIIZ name string containing the dynamic link entry point name of the routine in the subsystem to receive control when any of the registered functions is called. The maximum length of the ASCIIZ string is 33 bytes including the terminating byte of zero.

`flFun1 (ULONG)` - input
: A bit mask where each bit identifies a video function being registered. The bit definitions are shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits.

| Bit | Registered function | Mask |
|---|---|---|
| 31 | `VioPrtScToggle` | (flFun1 & VR_VIOPRTSCTOGGLE) |
| 30 | `VioEndPopUp` | (flFun1 & VR_VIOENDPOPUP) |
| 29 | `VioPopUp` | (flFun1 & VR_VIOPOPUP) |
| 28 | `VioSavRedrawUndo` | (flFun1 & VR_VIOSAVREDRAWUNDO) |
| 27 | `VioSavRedrawWait` | (flFun1 & VR_VIOSAVREDRAWWAIT) |
| 26 | `VioScrUnLock` | (flFun1 & VR_VIOSCRUNLOCK) |
| 25 | `VioScrLock` | (flFun1 & VR_VIOSCRLOCK) |
| 24 | `VioPrtSc` | (flFun1 & VR_VIOPRTSC) |
| 23 | `VioGetAnsi` | (flFun1 & VR_VIOGETANSI) |
| 22 | `VioSetAnsi` | (flFun1 & VR_VIOSETANSI) |
| 21 | `VioScrollRt` | (flFun1 & VR_VIOSCROLLRT) |
| 20 | `VioScrollLf` | (flFun1 & VR_VIOSCROLLLF) |
| 19 | `VioScrollDn` | (flFun1 & VR_VIOSCROLLDN) |

| | | |
|---|---|---|
| 18 | VioScrollUp | (flFun1 & VR_VIOSCROLLUP) |
| 17 | VioWrtCellStr | (flFun1 & VR_VIOWRTCELLSTR) |
| 16 | VioWrtCharStrAtt | (flFun1 & VR_VIOWRTCHARSTRATT) |
| 15 | VioWrtCharStr | (flFun1 & VR_VIOWRTCHARSTR) |
| 14 | VioWrtTTY | (flFun1 & VR_VIOWRTTTY) |
| 13 | VioWrtNCell | (flFun1 & VR_VIOWRTNCELL) |
| 12 | VioWrtNAttr | (flFun1 & VR_VIOWRTNATTR) |
| 11 | VioWrtNChar | (flFun1 & VR_VIOWRTNCHAR) |
| 10 | VioReadCellStr | (flFun1 & VR_VIOREADCELLSTR) |
| 9 | VioReadCharStr | (flFun1 & VR_VIOREADCHARSTR) |
| 8 | VioShowBuf | (flFun1 & VR_VIOSHOWBUF) |
| 7 | VioSetMode | (flFun1 & VR_VIOSETMODE) |
| 6 | VioSetCurType | (flFun1 & VR_VIOSETCURTYPE) |
| 5 | VioSetCurPos | (flFun1 & VR_VIOSETCURPOS) |
| 4 | VioGetPhysBuf | (flFun1 & VR_VIOGETPHYSBUF) |
| 3 | VioGetBuf | (flFun1 & VR_VIOGETBUF) |
| 2 | VioGetMode | (flFun1 & VR_VIOGETMODE) |
| 1 | VioGetCurType | (flFun1 & VR_VIOGETCURTYPE) |
| 0 | VioGetCurPos | (flFun1 & VR_VIOGETCURPOS) |

`flFun2 (ULONG)` - input
A bit mask where each bit identifies a video function being registered. The bit mask has the format shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits. Unused bits are reserved and must be set to zero.

| Bit | Registered function | Mask |
|---|---|---|
| 31–11 | Reserved, must be set to zero. | |
| 10 | VioDeRegister | (flFun2 & VR_VIODEREGISTER) |
| 9 | VioRegister | (flFun2 & VR_VIOREGISTER) |
| 8 | VioSetState | (flFun2 & VR_VIOSETSTATE) |
| 7 | VioGetState | (flFun2 & VR_VIOGETSTATE) |
| 6 | VioSetFont | (flFun2 & VR_VIOSETFONT) |
| 5 | VioGetCp | (flFun2 & VR_VIOGETCP) |
| 4 | VioSetCp | (flFun2 & VR_VIOSETCP) |
| 3 | VioGetConfig | (flFun2 & VR_VIOGETCONFIG) |
| 2 | VioGetFont | (flFun2 & VR_VIOGETFONT) |
| 1 | VioModeUndo | (flFun2 & VR_VIOMODEUNDO) |
| 0 | VioModeWait | (flFun2 & VR_VIOMODEWAIT) |

`usReturn (LONG)` - input
Reserved and must be zero.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 349 | ERROR_VIO_INVALID_MASK |

403 ERROR_VIO_INVALID_ASCIIZ
426 ERROR_VIO_REGISTER
494 ERROR_VIO_EXTENDED_SG

## Remarks

Notification of VIO calls issued within the hard error handler and DOS (real mode) sessions is not provided.

When control is routed to `EntryPoint`, the stack appears as it did after the original VIO call except that four additional values have been pushed onto the stack. The first is the index number (WORD) of the routine called. The second is a near pointer (WORD). The third is the caller's DS register (WORD). The fourth is the return address (DWORD) to the VIO router.

For example, if `VioSetCurPos` were a registered function, the stack would appear as if the following instruction sequence were executed if `VioSetCurPos` were called and control routed to `EntryPoint`:

```
PUSH WORD  Row
PUSH WORD  Column
PUSH WORD  hvio
CALL FAR   VioSetCurPos
PUSH WORD  Index
CALL NEAR  Entry point in Vio router
PUSH WORD  Caller's DS
CALL FAR   Dynamic link entry point
```

The index numbers that correspond to the registered functions are listed below:

| | | | |
|---|---|---|---|
| 0 | VioGetPhysBuf | 16 | VioWrtCellStr |
| 1 | VioGetBuf | 17 | VioWrtTTY |
| 2 | VioShowBuf | 18 | VioScrollUp |
| 3 | VioGetCurPos | 19 | VioScrollDn |
| 4 | VioGetCurType | 20 | VioScrollLf |
| 5 | VioGetMode | 21 | VioScrollRt |
| 6 | VioSetCurPos | 22 | VioSetAnsi |
| 7 | VioSetCurType | 23 | VioGetAnsi |
| 8 | VioSetMode | 24 | VioPrtSc |
| 9 | VioReadCharStr | 25 | VioScrLock |
| 10 | VioReadCellStr | 26 | VioScrUnLock |
| 11 | VioWrtNChar | 27 | VioSavRedrawWait |
| 12 | VioWrtNAttr | 28 | VioSavRedrawUndo |
| 13 | VioWrtNCell | 29 | VioPopUp |
| 14 | VioWrtCharStr | 30 | VioEndPopUp |
| 15 | VioWrtCharStrAtt | 31 | VioPrtScToggle |

32 VioModeWait
33 VioModeUndo
34 VioGetFont
35 VioGetConfig
36 VioSetCp
37 VioGetCp
38 VioSetFont
39 VioGetState
40 VioSetState
41 VioRegister
42 VioDeRegister

On entry to the global subsystem, AX contains the return code that is returned to the application that issued the VIO call. The global subsystem must return with all stack parameters and all general purpose registers, including AX, restored to the same values as on entry.

All VIO functions within a session are serialized on a thread basis. That is, when a global subsystem receives control, it can safely assume that it is not called again from the same session until the current call has completed. Note, however, that VIO calls across different sessions are not serialized.

VioGlobalReg may only be issued during system initialization. After system initialization, VioGlobalReg returns ERROR_VIO_REGISTER. A globally registered subsystem is active for the life of the system.

If multiple global subsystems are registered, they are given control in the order that they are registered.

A globally registered subsystem receives control on VIO calls issued from all full-screen sessions except the hard error handler and DOS (real mode) sessions.

# VioModeUndo (xWPM)

## Description

This call allows one thread within a process to cancel a `VioModeWait` issued by another thread within the same process.

```
#define INCL_VIO
#include <os2.h>

USHORT VioModeUndo (USHORT usOwnerInd, USHORT usKillInd,
                    USHORT usReserved);
```

`usOwnerInd (USHORT)` - input

Indicates whether the thread issuing VioModeUndo wants ownership of `VioModeWait` to be reserved for its process.

| Value | Definition |
|---|---|
| 0 | Reserve ownership<br>(usOwnerInd == UNDOI_GETOWNER) |
| 1 | Give up ownership<br>(usOwnerInd == UNDOI_RELEASEOWNER) |

`usKillInd (USHORT)` - input

Indicates whether the thread (with the outstanding `VioModeWait`) should be returned an error code or be terminated.

| Value | Definition |
|---|---|
| 0 | Return error code<br>(usKillInd == UNDOK_ERRORCODE) |
| 1 | Terminate thread.<br>(usKillInd == UNDOK_TERMINATE) |

`usReserved (USHORT)` - input

Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 421 | ERROR_VIO_INVALID_PARMS |
| 422 | ERROR_VIO_FUNCTION_OWNED |
| 427 | ERROR_VIO_NO_MODE_THREAD |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 465 | ERROR_VIO_DETACHED |
| 486 | ERROR_VIO_BAD_RESERVE |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

`VioModeUndo` may be issued only by a thread within the process that owns `VioModeWait`. The thread issuing VioModeUndo can either reserve ownership of the `VioModeWait` function for its process or give up ownership. The thread whose `VioModeWait` is cancelled is optionally terminated.

# VioModeWait (xWPM)

## Description

This call allows a graphics mode application to be notified when it must restore its video mode, state, and modified display adapter registers. The return from this function call provides the notification.

```
#define INCL_VIO
#include <os2.h>

USHORT VioModeWait (USHORT usReqType, USHORT pNotifyType,
                    USHORT usReserved);
```

`usReqType (USHORT)` - input
: Application request event. `RequestType` = 0 indicates that the application wants to be notified at the end of a pop-up to restore its mode. `RequestType` = 0 is the only event supported by `VioModeWait`.

`pNotifyType (PUSHORT)` - output
: Address of the operation to be performed by the application returning from `VioModeWait`. `NotifyType` = 0, indicating restore mode, is the only type of notification returned.

`usReserved (USHORT)` - input
: Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 421 | ERROR_VIO_INVALID_PARMS |
| 422 | ERROR_VIO_FUNCTION_OWNED |
| 423 | ERROR_VIO_RETURN |
| 424 | ERROR_SCS_INVALID_FUNCTION |
| 428 | ERROR_VIO_NO_SAVE_RESTORE_THD |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

At the completion of an application or hard error pop-up (reference `VioPopUp`), OS/2 notifies the session that was originally interrupted for the pop-up to restore its mode. The return from this function call provides that notification. The thread that issued the call must perform the restore and then immediately reissue `VioModeWait`.

When an application's VioModeWait thread is notified, the thread must restore its video mode, state, and modified display adapter registers. An application's VioModeWait thread does not restore the physical display buffer. OS/2 saves/restores the physical display buffer over a pop-up.

Only one process for a session can issue VioModeWait. The first process that issues VioModeWait becomes the owner of this function. (Refer to VioModeUndo.)

An application must issue VioModeWait only if it writes directly to the registers on the display adapter. Otherwise, the application can allow OS/2 to perform the required restore by not issuing VioModeWait.

When an application issues VioModeWait, it is also required to issue VioSavRedrawWait to be notified at screen switch time to perform a full save or restore (reference VioSavRedrawWait.) Two application threads must be dedicated to performing these operations.

# VioPopUp (xPM)

## Description

This call is issued by an application process when it requires a temporary screen to display a momentary message to the user.

```
#define INCL_VIO
#include <os2.h>

USHORT VioPopUp (PUSHORT pflags, HVIO hvio);
```

`pflags (PUSHORT)` - input

Address of the bit flags that indicates which options to the application are being selected.

| Bit | Description |
|---|---|
| 15–2 | Reserved, set to zero. |
| 1 | 0 = Non-transparent operation. The video mode is set to text-mode 3, 3*, 3+, 7, or 7+. The highest resolution supported by the primary display adapter configured in the system is selected. The screen is cleared, and the cursor is positioned in the upper left corner of the display.<br>!(*pflags & VP_TRANSPARENT)<br>1 = Transparent operation. If the video mode of the outgoing foreground session is text (mode 2, 3, 7, or one of the * or + variations of these modes), no mode change occurs. The screen is not cleared, and the cursor remains in its current position. If transparent operation is selected, and if the video mode of the outgoing foreground session is not text (or if the outgoing foreground session has a `VioSavRedrawWait` thread), the pop-up request is refused. A unique error code ERROR_VIO_TRANSPARENT_POPUP is returned in this case.<br>(*pflags & VP_TRANSPARENT)<br>OS/2 is responsible for saving and restoring the physical display buffer of the previous foreground session around a pop-up. This is true whether transparent or non-transparent operation is selected. |
| 0 | 0 = Return with unique error code ERROR_VIO_EXISTING_POPUP if pop-up is not immediately available.<br>!(*pflags & VP_WAIT)<br>1 = Wait if pop-up is not immediately available.<br>(*pflags & VP_WAIT) |

`hvio (HVIO)` - input

Reserved words of 0s.

## Return value

0 NO_ERROR
405 ERROR_VIO_NO_POPUP
406 ERROR_VIO_EXISTING_POPUP
483 ERROR_VIO_TRANSPARENT_POPUP

## Remarks

`VioPopUp` is normally issued by the application when it is running in the background and wishes to immediately display a message to the user without waiting to become the active foreground session.

When an application process issues `VioPopUp`, it should wait for the return from the request. If the process allows any of its threads to write to the screen before `VioPopUp` returns a successful return code, the screen output may be directed to the application's normal video buffer rather than to the pop-up screen. If the process allows any of its threads to issue keyboard or mouse calls before `VioPopUp` returns a successful return code, the input is directed from the application's normal session. Once the process that issued `VioPopUp` receives a successful return code, video and keyboard calls issued by any of the threads in the pop-up process are directed to the pop-up screen. This continues until the process issues `VioEndPopUp`. At that time video and keyboard calls resume being directed to the application's normal video buffer.

There may be only one pop-up in existence at any time. If a process requests a pop-up and a pop-up already exists, the process has the choice of waiting for the prior pop-up to complete or receiving an immediate return with an error code. The error code indicates that the operation failed due to an existing pop-up having captured the screen.

Video pop-ups provide a mechanism for a background application to notify the operator of an abnormal event upon which the operator must take some action. When considering the suitability of using pop-ups in a particular situation, the possible disruptive effect of pop-ups to the operator should be considered. If the operator were interrupted frequently by pop-ups issued by background applications,the operator would not effectively work with the foreground application.

While a video pop-up is in the foreground, the operator cannot use the hot key to switch to another application or to the shell. Before the operator can switch another application or the shell to the foreground, the pop-up application must issue `VioEndPopUp`.

While a video pop-up is in effect, all video calls from the previous foreground session are blocked until the process that issued `VioPopUp` issues `VioEndPopUp`.

When VioPopUp is issued, only the process within the session that issued VioPopUp is brought to the foreground. Assuming the session was already the foreground session, any video calls issued by other processes in that session are blocked until the process that issued VioPopUp issues VioEndPopUp.

DosExecPgm may not be issued by a process during a pop-up. The following video calls are the only calls that may be issued during the pop-up by the process that issued VioPopUp:

```
VioEndPopUp      VioScrollLf
VioGetConfig     VioSetCurPos
VioGetCp         VioSetCurType
VioGetFont       VioSetCp
VioGetAnsi       VioSetFont
VioGetState      VioSetState
VioGetCurPos     VioWrtNChar
VioGetCurType    VioWrtNAttr
VioGetMode       VioWrtNCell
VioReadCharStr   VioWrtCharStr
VioReadCellStr   VioWrtCharStrAtt
VioScrollRt      VioWrtCellStr
VioScrollUp      VioWrtTTY
VioScrollDn
```

Selectors to the physical display buffer that the issuing process obtained on a prior VioGetPhysBuf call may not be used during the pop-up.

When an application registers a replacement for VioPopUp within a session, the registered routine is invoked only when that session is in the foreground. If VioPopUp is issued when that session is in the background, the OS/2 default routine is invoked. If the application's session is using a keyboard or mouse monitor, the monitor does not intercept data while the pop-up is active.

## PM considerations

This function can be used from within a PM application. Kbdxxx, Mouxxx, and Vioxxx calls (with a zero handle) are all allowed between VioPopUp and VioEndPopUp, and are directed to the pop-up screen. An error is returned if issued with a non-zero handle.

## Description

This call is issued by the Session Manager when the operator presses PrtSc.

```
#define INCL_VIO
#include <os2.h>

USHORT VioPrtSc (HVIO hvio);
```

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
402 ERROR_VIO_SMG_ONLY
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`VioPrtSc` supports text modes 0 through 3, and 7. An Alternate Video Subsystem may want to register a replacement for `VioPrtSc`. An advanced video subsystem could set a graphics mode while the mode known to the base video subsystem PrtSc routine is text. Then, if the operator presses PrtSc, the printer output is unpredictable. `VioPrtSc` is reserved for use by the Session Manager. Application programs may not issue `VioPrtSc`.

Three beeps are generated if a hard error is detected while writing to the printer.

# VioPrtScToggle (xWPM)

## Description

This call is called by the Session Manager when the operator presses Ctrl and PrtSc.

```
#define INCL_VIO
include <os2.h>

USHORT VioPrtScToggle (HVIO hvio);
```

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 402 | ERROR_VIO_SMG_ONLY |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

`VioPrtScToggle` toggles the Ctrl and PrtSc state of the foreground session. When the Ctrl and PrtSc state is on, all `VioWrtTTY` calls from that session are echoed to the print device.

`VioPrtScToggle` can only be called by the Session Manager. If an application issues this call, it receives an error code.

Three beeps are generated if a hard error is detected while writing to the printer. When Ctrl and PrtSc is turned off, the operator may have to press the Enter key to cause output spooled while Ctrl and PrtSc was active to be printed. This is because the spool files are closed when the next `VioWrtTTY` is executed in the session, such as writing the prompt in response to the Enter key.

## Description

This call reads a string of character-attribute pairs (cells) from the screen, starting at the specified location.

```
#define INCL_VIO
#include <os2.h>

USHORT VioReadCellStr (PCH pchCellStr, PUSHORT pcb, USHORT usRow,
                       USHORT usColumn, HVIO hvio);
```

`pchCellStr (PCH)` - output
Address of the buffer where the cell string is returned.

`pcb (PUSHORT)` - input/output
Address of the buffer length in bytes. It must take into account that each character-attribute(s) entry in the buffer is 2 or 4 bytes. If the length of the buffer is not sufficient, the last entry is not complete.

`usRow (USHORT)` - input
Starting row of the field to read, 0 is the top row.

`usColumn (USHORT)` - input
Starting column of the field to read, 0 is the leftmost column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 358 | ERROR_VIO_ROW |
| 359 | ERROR_VIO_COL |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a string read comes to the end of the line and is not complete, the string read continues at the beginning of the next line. If the read comes to the

end of the screen and is not complete, the read terminates and the length is set to the length of the buffer that was filled.

## PM considerations

`VioReadCellStr` reads a string of character/attributes (or cells) from the Advanced VIO presentation space starting at the specified location.

# VioReadCharStr (FAPI)

## Description

This call reads a string of characters from the display starting at the specified location.

```
#define INCL_VIO
#include <os2.h>

USHORT VioReadCharStr (PCH pchCellStr, PUSHORT pcb, USHORT usRow,
                       USHORT usColumn, HVIO hvio);
```

`pchCellstr (PCH)` - output
Address of the buffer where the character string is returned.

`pcb (PUSHORT)` - input/output
Address of the buffer length in bytes.

`usRow (USHORT)` - input
Starting row of the field to read, 0 is the top row.

`usColumn (USHORT)` - input
Starting column of the field to read, 0 is the leftmost column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 358 | ERROR_VIO_ROW |
| 359 | ERROR_VIO_COL |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a string read comes to the end of the line and is not complete, then the string read continues at the beginning of the next line. If the read comes to

the end of the screen and is not complete, the read terminates and the length is set to the number of characters read.

## PM considerations

`VioReadCharStr` reads a character string from the Advanced VIO presentation space starting at the specified location.

# VioRegister (xWPM)

## Description

This call registers an alternate video subsystem within a session.

```
#define INCL_VIO
#include <os2.h>

USHORT VioRegister (PSZ pszModName, PSZ pszEntryName,
                    ULONG flFun1, ULONG flFun2);
```

`pszModName (PSZ)` - input

Address of the ASCIIZ string containing the 1–8 character filename of the subsystem. The maximum length of the ASCIIZ string is 9 bytes including the terminating byte of zero. The module must be a dynamic link library but the name supplied must not include the .DLL extension.

`pszEntryName (PSZ)` - input

Address of the ASCIIZ name string containing the dynamic link entry point name of the routine in the subsystem to receive control when any of the registered functions is called. The maximum length of the ASCIIZ string is 33 bytes including the terminating byte of zero.

`flFun1 (ULONG)` - input

A bit mask where each bit identifies a video function being registered. The bit definitions are shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits.

| Bit | Registered function | Mask |
|---|---|---|
| 31 | VioPrtScToggle | (flFun1 & VR_VIOPRTSCTOGGLE) |
| 30 | VioEndPopUp | (flFun1 & VR_VIOENDPOPUP) |
| 29 | VioPopUp | (flFun1 & VR_VIOPOPUP) |
| 28 | VioSavRedrawUndo | (flFun1 & VR_VIOSAVREDRAWUNDO) |
| 27 | VioSavRedrawWait | (flFun1 & VR_VIOSAVREDRAWWAIT) |
| 26 | VioScrUnLock | (flFun1 & VR_VIOSCRUNLOCK) |
| 25 | VioScrLock | (flFun1 & VR_VIOSCRLOCK) |
| 24 | VioPrtSc | (flFun1 & VR_VIOPRTSC) |
| 23 | VioGetAnsi | (flFun1 & VR_VIOGETANSI) |
| 22 | VioSetAnsi | (flFun1 & VR_VIOSETANSI) |
| 21 | VioScrollRt | (flFun1 & VR_VIOSCROLLRT) |
| 20 | VioScrollLf | (flFun1 & VR_VIOSCROLLLF) |
| 19 | VioScrollDn | (flFun1 & VR_VIOSCROLLDN) |
| 18 | VioScrollUp | (flFun1 & VR_VIOSCROLLUP) |

| Bit | Registered function | Mask |
|---|---|---|
| 17 | VioWrtCellStr | (flFun1 & VR_VIOWRTCELLSTR) |
| 16 | VioWrtCharStrAtt | (flFun1 & VR_VIOWRTCHARSTRATT) |
| 15 | VioWrtCharStr | (flFun1 & VR_VIOWRTCHARSTR) |
| 14 | VioWrtTTY | (flFun1 & VR_VIOWRTTTY) |
| 13 | VioWrtNCell | (flFun1 & VR_VIOWRTNCELL) |
| 12 | VioWrtNAttr | (flFun1 & VR_VIOWRTNATTR) |
| 11 | VioWrtNChar | (flFun1 & VR_VIOWRTNCHAR) |
| 10 | VioReadCellStr | (flFun1 & VR_VIOREADCELLSTR) |
| 9 | VioReadCharStr | (flFun1 & VR_VIOREADCHARSTR) |
| 8 | VioShowBuf | (flFun1 & VR_VIOSHOWBUF) |
| 7 | VioSetMode | (flFun1 & VR_VIOSETMODE) |
| 6 | VioSetCurType | (flFun1 & VR_VIOSETCURTYPE) |
| 5 | VioSetCurPos | (flFun1 & VR_VIOSETCURPOS) |
| 4 | VioGetPhysBuf | (flFun1 & VR_VIOGETPHYSBUF) |
| 3 | VioGetBuf | (flFun1 & VR_VIOGETBUF) |
| 2 | VioGetMode | (flFun1 & VR_VIOGETMODE) |
| 1 | VioGetCurType | (flFun1 & VR_VIOGETCURTYPE) |
| 0 | VioGetCurPos | (flFun1 & VR_VIOGETCURPOS) |

`flFun2 (ULONG)` - input

A bit mask where each bit identifies a video function being registered. The bit mask has the format shown below. The first word pushed onto the stack contains the high-order 16 bits of the function mask, and the second word contains the low-order 16 bits. Unused bits are reserved and must be set to zero.

| Bit | Registered function | Mask |
|---|---|---|
| 31–11 | Reserved, must be set to zero. | |
| 10 | VioDeRegister | (flFun2 & VR_VIODEREGISTER) |
| 9 | VioRegister | (flFun2 & VR_VIOREGISTER) |
| 8 | VioSetState | (flFun2 & VR_VIOSETSTATE) |
| 7 | VioGetState | (flFun2 & VR_VIOGETSTATE) |
| 6 | VioSetFont | (flFun2 & VR_VIOSETFONT) |
| 5 | VioGetCp | (flFun2 & VR_VIOGETCP) |
| 4 | VioSetCp | (flFun2 & VR_VIOSETCP) |
| 3 | VioGetConfig | (flFun2 & VR_VIOGETCONFIG) |
| 2 | VioGetFont | (flFun2 & VR_VIOGETFONT) |
| 1 | VioModeUndo | (flFun2 & VR_VIOMODEUNDO) |
| 0 | VioModeWait | (flFun2 & VR_VIOMODEWAIT) |

## Return value

0 NO_ERROR
349 ERROR_VIO_INVALID_MASK
403 ERROR_VIO_INVALID_ASCIIZ

426 ERROR_VIO_REGISTER
430 ERROR_VIO_ILLEGAL_DURING_POPUP
465 ERROR_VIO_DETACHED
494 ERROR_VIO_EXTENDED_SG

## Remarks

An alternate video subsystem must register which video calls it handles. The default OS/2 video subsystem is the Base Video Subsystem.

When any of the registered functions are called, control is then routed to `EntryPoint`. When this routine is entered, four additional values (5 words) are pushed onto the stack.

The first value is the index number (Word) of the routine being called. The second value is a near pointer (Word). The third value is the caller's DS register (Word). The fourth value is the return address (DWord) to the VIO router.

For example, if `VioSetCurPos` were a registered function, the stack would appear as if the following instruction sequence were executed if `VioSetCurPos` were called and control routed to `EntryPoint`:

```
PUSH WORD  Row
PUSH WORD  Column
PUSH WORD  hvio
CALL FAR   VioSetCurPos
PUSH WORD  Index
CALL NEAR  Entry point in Vio router
PUSH WORD  Caller's DS
CALL FAR   Dynamic link entry point
```

The index numbers that correspond to the registered functions are listed here:

```
 0 VioGetPhysBuf    13 VioWrtNCell
 1 VioGetBuf        14 VioWrtCharStr
 2 VioShowBuf       15 VioWrtCharStrAtt
 3 VioGetCurPos     16 VioWrtCellStr
 4 VioGetCurType    17 VioWrtTTY
 5 VioGetMode       18 VioScrollUp
 6 VioSetCurPos     19 VioScrollDn
 7 VioSetCurType    20 VioScrollLf
 8 VioSetMode       21 VioScrollRt
 9 VioReadCharStr   22 VioSetAnsi
10 VioReadCellStr   23 VioGetAnsi
11 VioWrtNChar      24 VioPrtSc
12 VioWrtNAttr      25 VioScrLock
```

```
26 VioScrUnLock       34 VioGetFont
27 VioSavRedrawWait   35 VioGetConfig
28 VioSavRedrawUndo   36 VioSetCp
29 VioPopUp           37 VioGetCp
30 VioEndPopUp        38 VioSetFont
31 VioPrtScToggle     39 VioGetState
32 VioModeWait        40 VioSetState
33 VioModeUndo
```

When a registered function returns to the video router, the return code is interpreted as follows:

**Return code = 0**
No error. Do not invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = 0.

**Return code = -1**
No error. Invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = return code from Base Video Subsystem.

**Return code = error (not 0 or -1)**
Do not invoke the corresponding Base Video Subsystem routine. Return to caller with Return code = error.

When an application registers a replacement for VioPopUp within a session, the registered routine is only invoked when that session is in the foreground. If `VioPopUp` is issued when that session is in the background, the OS/2 default routine is invoked.

An alternate video subsystem should be designed so the routines registered do not cause any hard errors when they are invoked. Otherwise, a system lockout occurs. Code and data segments of registered routines, that might be loaded from diskette, must be preloaded.

All VIO functions within a session are serialized on a thread basis. That is, when an alternate video subsystem receives control, it can safely assume that it is not called again from the same session until the current call has completed.

# VioSavRedrawUndo (xWPM)

## Description

This call allows one thread within a process to cancel a `VioSavRedrawWait` issued by another thread within the same process.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSavRedrawUndo (USHORT usOwnerInd, USHORT usKillInd,
                         USHORT usReserved);
```

`usOwnerInd (USHORT)` - **input**
Indicates whether the thread issuing VioSavRedrawUndo wants ownership of VioSavRedrawWait to be reserved for its process.

| Value | Definition |
|---|---|
| 0 | Reserve ownership<br>(usOwnerInd == UNDOI_GETOWNER) |
| 1 | Give up ownership<br>(usOwnerInd == UNDOI_RELEASEOWNER) |

`usKillInd (USHORT)` - **input**
Indicates whether the thread with the outstanding `VioSavRedrawWait` should be returned an error code or be terminated.

| Value | Definition |
|---|---|
| 0 | Return error code<br>(usKillInd == UNDOK_ERRORCODE) |
| 1 | Terminate thread<br>(usKillInd == UNDOK_TERMINATE) |

`usReserved (HVIO)` - **input**
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 421 | ERROR_VIO_INVALID_PARMS |
| 422 | ERROR_VIO_FUNCTION_OWNED |
| 428 | ERROR_VIO_NO_SAVE_RESTORE_THD |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

The issuing thread can reserve ownership of `VioSavRedrawWait` for its process or give it up. The thread whose `VioSavRedrawWait` was cancelled is optionally terminated. `VioSavRedrawUndo` may be issued only by a thread within the same process that owns `VioSavRedrawWait`.

# VioSavRedrawWait (xWPM)

## Description

This call notifies a graphics mode application when it must save or redraw its screen image.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSavRedrawWait (USHORT usRedrawInd, PUSHORT pNotifyType,
                         SHORT usReserved);
```

`usRedrawInd (USHORT)` - input
Indicates which events the application is waiting for:

| Value | Definition |
|---|---|
| 0 | The Session Manager notifies the application for both save and redraw operations. |
| 1 | The Session Manager notifies the application for redraw operations only. |

`pNotifyType (PUSHORT)` - output
Address that specifies the operation to be performed by the application upon return from `VioSavRedrawWait`:

| Value | Definition |
|---|---|
| 0 | Save screen image<br>(*pNotifyType == VSRWN_SAVE) |
| 1 | Restore screen image<br>(*pNotifyType == VSRWN_REDRAW) |

`usReserved (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 421 | ERROR_VIO_INVALID_PARMS |
| 422 | ERROR_VIO_FUNCTION_OWNED |
| 423 | ERROR_VIO_RETURN |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

OS/2 uses VioSavRedrawWait to notify a graphics mode application to save or restore its screen image at screen switch time. The application in the outgoing foreground session is notified to perform a save. The application in the incoming foreground session is notified to perform a restore. The application must perform the action requested and immediately reissue VioSavRedrawWait. When an application performs a save, it saves its physical display buffer, video mode, and any other information the application needs to completely redraw its screen at restore time.

Only one process per session can issue VioSavRedrawWait. The process that first issues VioSavRedrawWait becomes the owner of the function.

A text mode application must issue VioSavRedrawWait only if the application writes directly to the registers on the display adapter. Assuming VioSavRedrawWait is not issued by a text mode application, OS/2 performs the required saves and restores.

An application that issues VioSavRedrawWait might also need to issue VioModeWait. This would allow the application to be notified when it must restore its mode at the completion of an application or hard error pop-up. Refer to VioModeWait for more information. Two application threads would be required to perform these operations in this case.

At the time a VioSavRedrawWait thread is notified, the session is in transition to/from the background. Although the session's official status is background, any selector to the physical display buffer previously obtained by the VioSavRedrawWait process (through VioGetPhysBuf) is valid at this time. The physical display buffer must be accessed without issuing VioScrLock. Since the session's official status is background, any thread waits if it issues VioScrLock with the "wait if unsuccessful" option.

An application containing a VioSavRedrawWait thread should be designed so that the process does not cause any hard errors while the VioSavRedrawWait thread is running; otherwise a system lockout may occur.

An application's VioSavRedrawWait thread may be notified to perform a restore before it is notified to perform a save. This happens if the application was running in the background the first time it issued VioSavRedrawWait. The return from this function call provides the notification. The thread that issues the call performs the save or redraw and then reissues VioSavRedrawWait to wait until its screen image must be saved or redrawn again.

# VioScrLock (FAPI, xWPM)

## Description

This call requests ownership of (locks) the physical display buffer.

```
define INCL_VIO
#include <os2.h>

USHORT VioScrLock (USHORT fWait, PUCHAR pfNotLocked, HVIO hvio);
```

`fWait (USHORT)` - input
Indicates whether the process should block until the screen I/O can take place.

| Value | Definition |
|---|---|
| 0 | Return if screen I/O not available (fWait == VP_NOWAIT) |
| 1 | Wait until screen I/O is available (fWait == VP_WAIT) |

`pfNotLocked (PUCHAR)` - output
Address of the Indicator of whether the lock is successful, described here.

| Value | Definition |
|---|---|
| 0 | Lock successful (*pfNotLocked == LOCK_SUCCESS) |
| 1 | Lock unsuccessful (in the case of no wait) (*pfNotLocked == LOCK_FAIL) |

Status is returned only when AX = 0.

Status = 1 may be returned only when WaitFlag = 0.

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 366 | ERROR_VIO_WAIT_FLAG |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 434 | ERROR_VIO_LOCK |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

This function call permits a process to determine if I/O to the physical screen buffer can take place. This prevents the process from writing to the physical buffer when the process is in the background. Processes must co-operate with the system in coordinating screen accesses.

Screen switching is disabled while the screen lock is in place. If a screen switch is suspended by a screen lock, and if the application holding the lock does not issue `VioScrUnLock` within a system-defined time limit, the screen switch occurs, and the process holding the lock is frozen in the background. A process should yield the screen lock as soon as possible to avoid being frozen when running in the background. The timeout on the lock does not begin until a screen switch is requested.

When the screen lock is in effect and another thread in the same or different process (in the same session) issues `VioScrLock`, the second thread receives an error code. `VioScrUnLock` must be issued by a thread within the same process that issued `VioScrLock`.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to `VioScrLock` when coding in the DOS mode:

The status always indicates the lock is successful (Return code = 0).

# VioScrollDn (FAPI)

## Description

This call scrolls the entire display buffer (or area specified within the display buffer) down.

```
#define INCL_VIO
#include <os2.h>

USHORT VioScrollDn (USHORT usTopRow, USHORT usLeftCol,
                    USHORT usBotRow, USHORT usRightCol,
                    USHORT cbLines, PBYTE pCell, HVIO hvio);
```

`usTopRow (USHORT)` - input
Top row to be scrolled.

`usLeftCol (USHORT)` - input
Left column to be scrolled.

`usBotRow (USHORT)` - input
Bottom row to be scrolled.

`usRightCol (USHORT)` - input
Right column to be scrolled.

`cbLines (USHORT)` - input
Number of lines to be inserted at the top of the screen area being scrolled. If 0 is specified, no lines are scrolled.

`pCell (PBYTE)` - input
Address of the character-attribute(s) pair (2 or 4 bytes) used as a fill character on inserted lines.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`TopRow` = 0 and `LeftCol` = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for `TopRow`, `LeftCol`, `BotRow`, `RightCol`, or `Lines`, the maximum value for that parameter is used.

If `TopRow` and `LeftCol` = 0 and if `BotRow`, `RightCol`, and `Lines` = 65535 (or –1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

## Description

This call scrolls the entire display buffer (or area specified within the display buffer) to the left.

```
#define INCL_VIO
#include <os2.h>

USHORT VioScrollLf (USHORT usTopRow, USHORT usLeftCol,
                    USHORT usBotRow, USHORT usRightCol,
                    USHORT cbCol, PBYTE pCell, HVIO hvio);
```

`usTopRow (USHORT)` - input
Top row to be scrolled.

`usLeftCol (USHORT)` - input
Left column to be scrolled.

`usBotRow (USHORT)` - input
Bottom row to be scrolled.

`usRightCol (USHORT)` - input
Right column to be scrolled.

`cbLines (USHORT)` - input
Number of columns to be inserted at the right of the screen area being scrolled. If 0 is specified, no lines are scrolled.

`pCell (PBYTE)` - input
Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted columns.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`TopRow` = 0 and `LeftCol` = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for `TopRow`, `LeftCol`, `BotRow`, `RightCol`, or `Lines`, the maximum value for that parameter is used.

If `TopRow` and `LeftCol` = 0 and if `BotRow`, `RightCol`, and `Lines` = 65535 (or –1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

## Description

This call scrolls the entire display buffer (or area specified within the display buffer) to the right.

```
#define INCL_VIO
#include <os2.h>

USHORT VioScrollRt (USHORT usTopRow, USHORT usLeftCol,
                    USHORT usBotRow, USHORT usRightCol,
                    USHORT cbCol, PBYTE pCell, HVIO hvio);
```

`usTopRow (USHORT)` - input
Top row to be scrolled.

`usLeftCol (USHORT)` - input
Left column to be scrolled.

`usBotRow (USHORT)` - input
Bottom row to be scrolled.

`usRightCol (USHORT)` - input
Right column to be scrolled.

`cbCol (USHORT)` - input
Number of columns to be inserted at the left of the screen area being scrolled. If 0 is specified, no lines are scrolled.

`pCell (PBYTE)` - input
Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted columns.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`TopRow` = 0 and `LeftCol` = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for `TopRow`, `LeftCol`, `BotRow`, `RightCol`, or `Lines`, the maximum value for that parameter is used.

If `TopRow` and `LeftCol` = 0 and if `BotRow`, `RightCol`, and `Lines` = 65535 (or −1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

# VioScrollUp (FAPI)

## Description

This call scrolls the entire display buffer (or area specified within the display buffer) up.

```
#define INCL_VIO
#include <os2.h>

USHORT VioScrollUp (USHORT usTopRow, USHORT usLeftCol,
                    USHORT usBotRow, USHORT usRightCol,
                    USHORT cbLines, PBYTE pCell, HVIO hvio);
```

`usTopRow (USHORT)` - input
Top row to be scrolled.

`usLeftCol (USHORT)` - input
Left column to be scrolled.

`usBotRow (USHORT)` - input
Bottom row to be scrolled.

`usRightCol (USHORT)` - input
Right column to be scrolled.

`cbLines (USHORT)` - input
Number of lines to be inserted at the bottom of the screen area being scrolled. If 0 is specified, no lines are scrolled.

`pCell (PBYTE)` - input
Address of the character attribute(s) pair (2 or 4 bytes) used as a fill character on inserted lines.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`TopRow` = 0 and `LeftCol` = 0 identifies the top left corner of the screen.

If a value greater than the maximum value is specified for `TopRow`, `LeftCol`, `BotRow`, `RightCol`, or `Lines`, the maximum value for that parameter is used.

If `TopRow` and `LeftCol` = 0 and if `BotRow`, `RightCol`, and `Lines` = 65535 (or −1 in assembler language), the entire screen is filled with the character-attribute pair defined by Cell.

# VioScrUnLock (FAPI, xWPM)

## Description

This call releases ownership of (unlocks) the physical display buffer.

```
#define INCL_VIO
#include <os2.h>

USHORT VioScrUnLock (HVIO hvio);
```

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 367 | ERROR_VIO_UNLOCK |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

This call releases the screen lock that is set by `VioScrLock`. The `VioScrUnLock` call must be issued by a thread in the same process as the thread that issued `VioScrLock`.

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following restriction applies to `VioScrUnLock` when coding in the DOS mode:

The status always indicates the unlock is successful (return code = 0).

# VioSetAnsi (xPM)

## Description

This call activates or deactivates ANSI support.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetAnsi (USHORT fAnsi, HVIO hvio);
```

`fAnsi (USHORT)` - input
Equals 1 to activate ANSI support (fAnsi == ANSI_ON) or 0 (fAnsi == ANSI_OFF) to deactivate ANSI.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 421 | ERROR_VIO_INVALID_PARMS |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

For ANSI support, "ON" is the default.

## Description

This call allows a process to set the code page used to display text data on the screen for the specified handle.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetCp (USHORT usReserved, USHORT idCodePage, HVIO hvio);
```

`usReserved (USHORT)` - input
Reserved word of 0s.

`idCodePage (USHORT)` - input
The CodePageID must be either 0, –1, –2, or have been specified on the CONFIG.SYS CODEPAGE = statement. A value of 0000 indicates that the code page is to be set to the default ROM code page provided by the hardware. A value of –1 enables the user font codepage if user fonts have previously been set with `VioSetFont`. A value of –2 disables the user font and re-enables the prepared system codepage or ROM codepage that was active before the user font was enabled.

If the code page ID is not 0, –1, –2, or does not match one of the IDs on the CODEPAGE = statement, an error results. Refer to *IBM Operating System/2.0 Command Reference* for a complete description of CODEPAGE.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |
| 469 | ERROR_VIO_BAD_CP |
| 470 | ERROR_VIO_NO_CP |
| 471 | ERROR_VIO_NA_CP |

## Remarks

`VioSetCp` can be used to enable and disable the user font code page as well as the prepared system code pages. If a prepared system code page or the

ROM code page is specified, any previously set code page is disabled and the specified code page is enabled.

Specifying the special code page of –1 enables the user font code page if user fonts have previously been set. Specifying the special code page of –2 disables the user font code page and re-enables the prepared system code page or ROM code page that was active before the user font code page was enabled.

## PM considerations

Valid CodePageID values are either 0 or one that was specified on the CONFIG.SYS CODEPAGE = statement; –1 and –2 are not valid for PM.

This call can be used to set an EBCDIC code page for Advanced VIO. For a full-screen or Vio-windowed application, this function causes the displayed characters to be reinterpreted immediately in the new code page. For a Presentation Manager application, the characters in the base font are reinterpreted in the new code page only when other events cause the characters to be repainted. This function has no effect on displayed characters that use a font other than the base font.

# VioSetCurPos (FAPI)

## Description

This call sets the cursor's coordinates on the screen.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetCurPos (USHORT usRow, USHORT usColumn, HVIO hvio);
```

`usRow (USHORT)` - input
New cursor row position, 0 is the top row.

`usColumn (USHORT)` - input
New cursor column position, 0 is the leftmost column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

# VioSetCurType (FAPI)

## Description

This call sets the cursor type.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetCurType (PVIOCURSORINFO pvioCursorInfo, HVIO hvio);
```

`pvioCursorInfo (PVIOCURSORINFO)` - input
Address of the cursor characteristics structure:

`yStart (USHORT)`
Horizontal scan line in the character cell that marks the top line of the cursor. If the character cell has n scan lines, 0 is the top scan line of the character cell and (n – 1) is the bottom scan line.

`cEnd (USHORT)`
Horizontal scan line in the character cell that marks the bottom line of the cursor. Scan lines within a character cell are numbered as defined in startline. The maximum value allowed is 31.

`cx (USHORT)`
Width of the cursor. In text modes, cursorwidth is the number of columns. The maximum number supported by the OS/2 base video subsystem is 1. In graphics modes, cursorwidth is the number of pels.

A value of 0 specifies the default width. In text modes, this is 1 column. In graphics modes, this is the number of pels equivalent to the width of one character.

`attr (USHORT)`
A value of –1 denotes a hidden cursor, all other values denote a normal cursor.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 356 | ERROR_VIO_WIDTH |
| 421 | ERROR_VIO_INVALID_PARMS |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

To set `CursorStartLine` and `CursorEndLine` independent of the number of scan lines for each character cell, you may specify these parameters as percentages. OS/2 then calculates the physical start and end scan lines, respectively, by multiplying the percentage specified for the parameter by the total number of scan lines in the character cell and rounding to the nearest scan line. Percentages are specified as negative values (or 0) in the range 0 through –100. Specifying `CursorStartLine` = –90 and `CursorEndLine` = –100 requests a cursor that occupies the bottom 10 percent of the character cell.

## PM considerations

Set the cursor type. The cursor type consists of the cursor start line, end line, width (assumed 0 - one column width) and attribute (normal or hidden).

# VioSetFont (FAPI, xWPM)

## Description

This call downloads a display font. The font being set must be compatible with the current mode.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetFont (PVIOFONTINFO pviofi, HVIO hvio);
```

`pviofi (PVIOFONTINFO)` - input
Address of the font structure containing the request:

`cb (USHORT)`
Length of structure, including cb.

14 Only valid value.

`type (USHORT)`
Request type:

| Type | Definition |
|---|---|
| 0 | Set current RAM font for EGA, VGA, or IBM Personal System/2 Display Adapter. |

`cxCell (USHORT)`
Pel columns in character cell.

`cyCell (USHORT)`
Pel rows in character cell.

`pbData (PVOID)`
Address of the data area containing font table to set.

`cbData (USHORT)`
Length, in bytes, of the caller-supplied data area; must be 256 times the character cell height specified in pelrows.

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 421 | ERROR_VIO_INVALID_PARMS |
| 436 | ERROR_VIO_INVALID_HANDLE |

438 ERROR_VIO_INVALID_LENGTH
465 ERROR_VIO_DETACHED
467 ERROR_VIO_FONT
468 ERROR_VIO_USER_FONT
494 ERROR_VIO_EXTENDED_SG

## Remarks

`VioSetFont` is applicable only for the enhanced graphics adapter, VGA or IBM Personal System/2 Display Adapter.

**Note** Although graphics mode support is provided in `VioSetFont`, this support is not provided by the Base Video Handlers provided with OS/2.

When `VioSetFont` is issued, the current code page is reset. If `VioGetCp` is subsequently issued, the error code ERROR_VIO_USER_FONT is returned. Return code, ERROR_VIO_USER_FONT represents a warning. It indicates that although the font could not be loaded into the adapter using the current mode, the font was saved as part of a special user font code page for use with a later `VioSetMode`. Successfully setting a user font sets the special user font code page, just as if a code page of –1 was specified using `VioSetCp`.

The user font code page consists of the most recent user font of each size that was set by `VioSetFont`. For example, if two 8×12 fonts and three 8×16 fonts had been set, only two fonts, the most recent of the 8×12 and 8×16 fonts, would be saved.

The special code page is used in the same way as those code pages specified on the CODEPAGE = statement in CONFIG.SYS.

# VioSetMode (FAPI, xPM)

## Description

This call sets the mode of the display.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetMode (PVIOMODEINFO pvioModeInfo, HVIO hvio);
```

`pvioModeInfo (PVIOMODEINFO)` - input
Address of the mode characteristics structure:

`cb (USHORT)`
Input parameter to `VioSetMode`. Length specifies the cb of the data structure in bytes including Length itself. The minimum structure size required is 3 bytes. OS/2 sets to the first mode (in the list of modes supported by this display configuration) with a data structure matching the mode data specified.

`fbType (UCHAR)`
Mode characteristics bit mask:

| Bit | Description |
|---|---|
| 7–4 | Reserved, set to zero. |
| 3 | 0 = VGA-compatible modes 0 thru 13H.<br>1 = Native mode. |
| 2 | 0 = Enable color burst<br>!(fbType & VGMT_DISABLEBURST)<br>1 = Disable color burst<br>(fbType & VGMT_DISABLEBURST) |
| 1 | 0 = Text mode<br>!(fbType & VGMT_GRAPHICS)<br>1 = Graphics mode<br>(fbType & VGMT_GRAPHICS) |
| 0 | 0 = Monochrome compatible mode<br>!(fbType & VGMT_OTHER)<br>1 = Other.<br>(fbType & VGMT_OTHER) |

`color (UCHAR)`
Number of colors defined as a power of 2. This is equivalent to the number of color bits that define the color, for example:

| Value | Definition |
|---|---|
| 0 | Monochrome modes 7, 7+, and F. |
| 1 | 2 colors. |
| 2 | 4 colors. |
| 4 | 16 colors. |
| 8 | 256 colors. |

`col (USHORT)`
Number of text columns.

`row (USHORT)`
Number of text rows.

`hres (USHORT)`
Horizontal resolution, number of pel columns.

`vres (USHORT)`
Vertical resolution, number of pel rows.

`fmt_ID (UCHAR)`
Identifies the format of the attributes.

`attrib (UCHAR)`
Identifies the number of attributes in a character cell.

`buf_addr (ULONG)`
32-bit physical address of the physical display buffer for this mode.

`buf_length (ULONG)`
Length of the physical display buffer for this mode.

`full_length (ULONG)`
Size of the buffer required for a full save of the physical display buffer for this mode.

`partial_length (ULONG)`
Size of the buffer required for a partial (pop-up) save of the physical display buffer for this mode.

`ext_data_addr (PCH)`
Far address to an extended mode data structure or zero if none. The format of the extended mode data structure is determined by the device driver and is unknown to OS/2.

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 430 | ERROR_VIO_ILLEGAL_DURING_POPUP |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 438 | ERROR_VIO_INVALID_LENGTH |
| 465 | ERROR_VIO_DETACHED |
| 467 | ERROR_VIO_FONT |
| 468 | ERROR_VIO_USER_FONT |
| 494 | ERROR_VIO_EXTENDED_SG |

## Remarks

`VioSetMode` initializes the cursor position and type.

`VioSetMode` does not clear the screen. To clear the screen, use one of the VioScrollxx calls.

The disable color burst bit in the Type field in the `VioSetMode` data structure is functional only for the CGA and VGA. This bit causes the color portion of the video signal to be suppressed, producing a black and white mode on composite monitors attached to the CGA. On VGA, the bit causes the color lookup table to be loaded with values that produce shades of gray instead of colors, again producing a black and white mode. For all other combinations of adapters and displays, the setting of this bit is recorded and returned on any subsequent `VioGetMode` call, but otherwise is ignored.

For text modes in full-screen sessions, the number of rows on the screen is determined by the availability of fonts of the correct size. For any specified mode, the size of the character defined by the font must be (Horizontal Resolution)/(Text Columns) dots wide and (Vertical Resolution)/(Text Rows) dots high. For example, an 8×8 font would support 39 through 43 text rows if the screen resolution were 640×350.

If `VioSetState` request type 6 has been issued previously to select the target display configuration for `VioSetMode`, the mode is set on the display configuration selected. If that display configuration does not support the mode specified, an error is returned.

Assuming no target display configuration for VioSetMode is selected, the mode is set on the primary configuration. If the primary configuration does not support the mode specified, the mode is set on the secondary configuration.

The table below shows the `VioSetMode` parameters required to set all the modes supported by the CGA, EGA, VGA, and PS/2 Display Adapters. The

modes native to the 8514/A and other advanced video adapters are set with the Adapter (programming) Interface to these adapters, not `VioSetMode`.

**Note** Although graphics mode support is provided in `VioSetMode`, this support is not provided by the Base Video Handlers provided with OS/2.

**Table 3-2 Display mode attributes supported by adapters**

| Bios mode | Type | Color | ColS | Rows | Hres | Vres | Valid adapter/display combinations [emulated] |
|---|---|---|---|---|---|---|---|
| 0 | 5 | 4 | 40 | 25 | 320 | 200 | [CGA/CD],CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA Plasma |
| 0* | 5 | 4 | 40 | 25 | 320 | 350 | [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma |
| 0+ | 5 | 4 | 40 | 25 | 360 | 400 | VGA/Mono, VGA/Color |
| 0# | 5 | 4 | 40 | 25 | 320 | 400 | VGA/Mono, VGA/Color, VGA/Plasma |
| 1 | 1 | 4 | 40 | 25 | 320 | 200 | CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA Plasma] |
| 1* | 1 | 4 | 40 | 25 | 320 | 350 | EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma] |
| 1+ | 1 | 4 | 40 | 25 | 360 | 400 | [VGA/Mono], VGA/Color, |
| 1# | 1 | 4 | 40 | 25 | 320 | 400 | [VGA/Mono], VGA/Color, [VGA Plasma] |
| 2 | 5 | 4 | 80 | 25 | 640 | 200 | [CGA/CD], CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA Plasma |
| 2* | 5 | 4 | 80 | 25 | 640 | 350 | [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma |
| 2+ | 5 | 4 | 80 | 25 | 720 | 400 | VGA/Mono, VGA/Color |
| 2# | 5 | 4 | 80 | 25 | 640 | 400 | VGA/Mono, VGA/Color, VGA/Plasma |
| 3 | 1 | 4 | 80 | 25 | 640 | 200 | CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA,Plasma] |
| 3* | 1 | 4 | 80 | 25 | 640 | 350 | EGA/ECD, [VGA/Mono], VGA/Color, [VGA,Plasma] |

**Table 3-2 Continued**

| Bios mode | Type | Color | ColS | Rows | Hres | Vres | Valid adapter/display combinations [emulated] |
|---|---|---|---|---|---|---|---|
| 3+ | 1 | 4 | 80 | 25 | 720 | 400 | [VGA/Mono], VGA/Color |
| 3# | 1 | 4 | 80 | 25 | 640 | 400 | [VGA/Mono], VGA/Color, [VGA/Plasma] |
| 7 | 0 | 0 | 80 | 25 | 720 | 350 | MPA/MD, EGA/MD, VGA/Mono, VGA/Color |
| 7+ | 0 | 0 | 80 | 25 | 720 | 400 | VGA/Mono, VGA/Color |
| 7# | 0 | 0 | 80 | 25 | 640 | 400 | VGA/Mono, VGA/Color, VGA/Plasma |
| n/a | 0 | 0 | 80 | 25 | 640 | 350 | VGA/Mono, VGA/Color, VGA/Plasma |
| n/a | 1 | 4 | 80 | 30 | 720 | 480 | [VGA/Mono], VGA/Color |
| n/a | 1 | 4 | 80 | 30 | 640 | 480 | [VGA/Mono], VGA/Color, VGA/Plasma |
| 4 | 3 | 2 | [40] | [25] | 320 | 200 | CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma] |
| 5 | 7 | 2 | [40] | [25] | 320 | 200 | [CGA/CD], CGA/Comp, [EGA/CD], [EGA/ECD], VGA/Mono, VGA/Color, VGA/Plasma |
| 6 | 3 | 1 | [80] | [25] | 640 | 200 | CGA/CD, CGA/Comp, EGA/CD, EGA/ECD, VGA/Mono, VGA/Color, VGA/Plasma |
| D | 3 | 4 | [40] | [25] | 320 | 200 | EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma] |
| E | 3 | 4 | [80] | [25] | 640 | 200 | EGA/CD, EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma] |
| F | 2 | 0 | [80] | [25] | 640 | 350 | EGA/MD, VGA/Mono, VGA/Color, VGA/Plasma |
| 10 | 3 | 4 | [80] | [25] | 640 | 350 | EGA/ECD, [VGA/Mono], VGA/Color, [VGA/Plasma] |
| 11 | 3 | 1 | [80] | [30] | 640 | 480 | VGA/Mono, VGA/Color, VGA/Plasma |
| 12 | 3 | 4 | [80] | [30] | 640 | 480 | [VGA/Mono], VGA/Color, [VGA/Plasma] |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 13 | 3 | 8 | [40] | [25] | 320 | 200 | [VGA/Mono], VGA/Color, [VGA/Plasma] |
| n/a | 11 | 8 | [80] | [30] | 640 | 480 | [8514A/Mono], 8514A/Color |
| n/a | 11 | 4 | [80] | [30] | 640 | 480 | [8514A/Mono], 8514A/Color |
| n/a | 11 | 8 | [85] | [38] | 1024 | 768 | [8514A/HMono], 8514A/HColor |
| n/a | 11 | 4 | [85] | [38] | 1024 | 768 | [8514A/HMono], 8514A/HColor |

**Display adapters**

| | |
|---|---|
| MPA | Monochrome/Printer Adapter |
| CGA | Color Graphics Adapter |
| EGA | Enhanced Graphics Adapter |
| VGA | Video Graphics Array, PS/2 Display Adapter |
| 8514A | 8514A Display Adapter |

**Displays**

| | |
|---|---|
| MD | 5151 Monochrome Display |
| CD | 5153 Color Display |
| ECD | 5154 Enhanced Color Display |
| MONO | 8503 PS/2 Monochrome Display, 8507/8604 Display |
| HMONO | 8507/8604 Display |
| COLOR | 8512/13 PS/2 Color Display, 8514 Display |
| HCOLOR | 8514/Display |
| PLASMA | Plasma Display Panel |
| COMP | Composite Video Monitor |

**Notes**

Types 0, 1, and 5 are text modes; types 2, 3, 7 and 11 are graphics modes.

For BIOS mode, 0, 2, 5, the color burst is disabled on the CGA and VGA.

The Personal System/2 Display Adapter 8514/A (TM) has advanced function modes, which are supported through the 8514/A display adapter interface, not the VIO Subsystem. Refer to the *Personal System/2 Display Adapter 8514/A Technical Reference* for details of this support.

For text modes in full-screen, the number of rows might differ from the mode table due to the availability of fonts of the correct size as described earlier.

---

## PM considerations

Windowable VIO sessions support only 80-column, color text modes. When `VioSetMode` is called from a Windowable VIO session, it only verifies that an 80-column text mode was requested, with Text Rows between 1 and 255. The resulting mode, which can be queried using VioGetMode, always has Type = 1, Color = 4, Text Columns = 80, Text Rows = requested Text Rows, Horizontal Resolution = 640, and Vertical Resolution = 16 * (Text Rows).

## Family API considerations

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following consideration applies to `VioSetMode` when coding for the DOS mode:

`VioSetMode` clears the screen.

# VioSetState (FAPI, xWPM)

## Description

This call performs one of the following functions; set palette registers, sets the overscan (border) color, set the blink/background intensity switch, set color registers, set the underline location, or set the target `VioSetMode` display configuration.

```
#define INCL_VIO
#include <os2.h>

USHORT VioSetState (PVOID pState, HVIO hvio);
```

`pState (PVOID)` - input
Address of one of six different state structures, depending on the request type. The request type is the second word of the packet.

| Type | Definition |
|---|---|
| 0 | Set palette registers |
| 1 | Set overscan (border) color |
| 2 | Set blink/background intensity switch |
| 3 | Set color registers |
| 4 | Reserved |
| 5 | Set underline location |
| 6 | Set target VioSetMode display configuration |
| 7 | Reserved |

The six structures, depending on request type, are as follows:

**VIOPALSTATE**
Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
38 Maximum valid value.

`type (USHORT)` - input
Request type 0 for palette registers.

`iFirst (USHORT)` - input
First palette register in the palette register sequence; must be specified in the range 0 through 15. The palette registers are returned in sequential order. The number returned is based upon cb.

`acolor (USHORT[(cb-6)/2])` - input
Color value for each palette register. The maximum number of entries in the color value array is 16.

**VIOOVERSCAN**

Applies to CGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 1 for overscan (border) color.

`color (USHORT)` - input
Color value.

**VIOINTENSITY**

Applies to CGA, EGA, MCGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 2 for blink/background intensity switch.

`fs (USHORT)` - input
Switch set as:

| Value | Definition |
|---|---|
| 0 | Blinking foreground colors enabled. |
| 1 | High intensity background colors enabled. |

**VIOCOLORREG**

Applies to VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
12 Only valid value.

`type (USHORT)` - input
Request type 3 for color registers.

`firstcolorreg (USHORT)` - input
First color register to set in the color register sequence; must be specified in the range 0 through 255. The color registers are set in sequential order.

`numcolorregs (USHORT)` - input
Number of color registers to set; must be specified in the range 1 through 256.

`colorregaddr (PCH)` - input
Far address of a data area containing one three-byte entry for each color register to be set. The format of each entry is as follows:

Byte 1 Red value
Byte 2 Green value
Byte 3 Blue value.

**VIOSETULINELOC**

Applies to EGA, VGA, or IBM Personal System/2 Display Adapter.

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 5 to set the scan line for underlining. Underlining is enabled only when the foreground color is 1 or 9.

`scanline (USHORT)` - input
Scan line minus 1. Values of 0 through 31 are acceptable. A value of 32 means underlining is disabled.

**VIOSETTARGET**

`cb (USHORT)` - input
Length of structure, including cb.
6 Only valid value.

`type (USHORT)` - input
Request type 6 to set display configuration to be the target of the next VioSetMode.

`defaultalgorithm (USHORT)` - input
Configuration:

| Value | Definition |
|---|---|
| 0 | Default selection algorithm. See VioSetMode. (defaultalgorithm == VIO_CONFIG_CURRENT) |
| 1 | Primary (defaultalgorithm == VIO_CONFIG_PRIMARY) |
| 2 | Secondary. (defaultalgorithm == VIO_CONFIG_SECONDARY) |

`hvio (HVIO)` - input
Reserved word of 0s.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
421 ERROR_VIO_INVALID_PARMS
436 ERROR_VIO_INVALID_HANDLE
438 ERROR_VIO_INVALID_LENGTH
465 ERROR_VIO_DETACHED
494 ERROR_VIO_EXTENDED_SG

## Family API considerations

Request type = 6, Set Target `VioSetMode` Display Configuration, and request type = 5, Set Underline Location, are not supported in the Family API.

Some options operate differently in the DOS mode than in the OS/2 mode. Therefore, the following considerations applies to `VioSetMode` when coding for the DOS mode:

`VioSetMode` clears the screen.

# VioShowBuf (xPM)

## Description

This call updates the physical display buffer with the logical video buffer (LVB).

```
#define INCL_VIO
#include <os2.h>

USHORT VioShowBuf (USHORT offLVB, USHORT cb, HVIO hvio);
```

`offLVB (USHORT)` - input
Starting offset within the logical video buffer at which the update to the screen is to start.

`cb (USHORT)` - input
Length of the area to be updated to the screen.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
430 ERROR_VIO_ILLEGAL_DURING_POPUP
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

`VioShowBuf` is ignored unless it is issued by a process that has previously called `VioGetBuf` and that is currently executing in the foreground.

## PM considerations

This function updates the display with the Advanced VIO presentation space.

# VioWrtCellStr (FAPI)

## Description

This call writes a string of character-attribute pairs (cells) to the display.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtCellStr (PCH pchCellStr, USHORT cb,
                      USHORT usRow, USHORT usColumn, HVIO hvio);
```

`pchCellStr (PCH)` - input
Address of the string of character-attribute(s) cells to be written.

`cb (USHORT)` - input
Length, in bytes, of the string to be written. Each character-attribute(s) cell is 2 or 4 bytes.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 358 | ERROR_VIO_ROW |
| 359 | ERROR_VIO_COL |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write a character-attribute string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

## Description

This call writes a character string to the display.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtCharStrAtt (PCH pch, USHORT cb,
                         USHORT usRow, USHORT usColumn,
                         PBYTE pAttr, HVIO hvio);
```

`pch (PCH)` - input
Address of the character string to be written.

`cb (USHORT)` - input
Length, in bytes, of the character string.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write a character string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

# VioWrtCharStrAtt (FAPI)

## Description

This call writes a character string with repeated attribute to the display.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtCharStrAtt (PCH pch, USHORT cb, USHORT usRow,
                         USHORT usColumn, PBYTE pAttr, HVIO hvio);
```

`pch (PCH)` - input
Address of the character string to be written.

`cb (USHORT)` - input
Length, in bytes, of the character string.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`pAttr (PBYTE)` - input
Address of the attribute(s) (1 or 3 bytes) to be used in the display buffer for each character of the string written.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 358 | ERROR_VIO_ROW |
| 359 | ERROR_VIO_COL |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write a character string with a repeated attribute string to the Advanced VIO presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

# VioWrtNAttr (FAPI)

## Description

This call writes an attribute to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtNAttr (PBYTE pAttr, USHORT cb, USHORT usRow,
                    USHORT usColumn, HVIO hvio);
```

`pAttr (PBYTE)` - input
Address of the attribute(s) (1 or 3 bytes) to be written.

`cb (USHORT)` - input
Number of times to write the attribute.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write an attribute code to the Advanced VIO presentation space a specified number of times. The caller must specify the starting location on the presentation space where the string is to be written.

## Description

This call writes a cell (character-attribute pair) to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtNCell (PBYTE pCell, USHORT cb,
                    USHORT usRow, USHORT usColumn,
                    HVIO hvio);
```

`pCell (PBYTE)` - input
Address of the character-attribute(s) cell (2 or 4 bytes) to be written.

`cb (USHORT)` - input
Number of times to write the cell.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 358 | ERROR_VIO_ROW |
| 359 | ERROR_VIO_COL |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write a cell (character-attribute) to the Advanced VIO presentation space a specified number of times. The caller must specify the starting location on the presentation space where the string is to be written.

# VioWrtNChar (FAPI)

## Description

`VioWrtNChar` writes a character to the display a specified number of times.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtNChar (PCH pchChar, USHORT cb, USHORT usRow,
                    USHORT usColumn, HVIO hvio);
```

`pchChar (PCH)` - input
Address of the character to be written.

`cb (USHORT)` - input
Number of times to write the character.

`usRow (USHORT)` - input
Starting cursor row.

`usColumn (USHORT)` - input
Starting cursor column.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by `VioGetPs`.

## Return value

0 NO_ERROR
355 ERROR_VIO_MODE
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
465 ERROR_VIO_DETACHED

## Remarks

If a repeated write gets to the end of the line and is not complete, the write continues at the beginning of the next line. If the write gets to the end of the screen, the write terminates.

## PM considerations

Write a character to the Advanced VIO presentation space a number of times. The caller must specify the starting location on the presentation space where the string is to be written.

## Description

This call writes a character string to the display starting at the current cursor position. At the completion of the write, the cursor is positioned at the first position beyond the end of the string.

```
#define INCL_VIO
#include <os2.h>

USHORT VioWrtTTY (PCH pch, USHORT cb, HVIO hvio);
```

`pch (PCH)` - input
Address of the string to be written.

`cb (USHORT)` - input
Length of the character string in bytes.

`hvio (HVIO)` - input
This must be zero unless the caller is a Presentation Manager application, in which case it must be the value returned by VioGetPs.

## Return value

| | |
|---|---|
| 0 | NO_ERROR |
| 355 | ERROR_VIO_MODE |
| 436 | ERROR_VIO_INVALID_HANDLE |
| 465 | ERROR_VIO_DETACHED |

## Remarks

If a string write gets to the end of the line and is not complete, the string write continues at the beginning of the next line. If the write gets to the end of the screen, the screen is scrolled, and the write continues until completed.

The characters carriage return, line feed, backspace, tab, and bell are treated as commands rather than printable characters. Backspace is a non-destructive backspace. Tabs are expanded to provide standard 8-byte-wide fields. `VioWrtTTY` is the only video call affected by Ctrl-PrtSc and ANSI.

Characters are written using the current attribute defined by ANSI or the default value 7.

`VioWrtTTY` is supported in graphics mode to process ANSI sequences. This allows the application to enter and exit a graphics mode.

## PM considerations

Write a character string from the current cursor position in TTY mode to the Advanced VIO presentation space. The cursor is positioned after the last character written at the end of the write.

# Errors returned from base OS/2 calls

This appendix contains the error number, name, and description of the errors returned from the KBD/MOU/VIO calls.

0 NO_ERROR
No error occurred.

1 ERROR_INVALID_FUNCTION
Invalid function number.

2 ERROR_FILE_NOT_FOUND
File not found.

3 ERROR_PATH_NOT_FOUND
Path not found.

4 ERROR_TOO_MANY_OPEN_FILES
Too many open files (no handles left).

5 ERROR_ACCESS_DENIED
Access denied.

6 ERROR_INVALID_HANDLE
Invalid handle.

7 ERROR_ARENA_TRASHED
Memory control blocks destroyed.

8 ERROR_NOT_ENOUGH_MEMORY
Insufficient memory.

9 ERROR_INVALID_BLOCK
Invalid memory-block address.

10 ERROR_BAD_ENVIRONMENT
Invalid environment.

11 ERROR_BAD_FORMAT
Invalid format.

12 ERROR_INVALID_ACCESS
Invalid access code.

13 ERROR_INVALID_DATA
Invalid data.

14 Reserved.

15 ERROR_INVALID_DRIVE
Invalid drive specified.

16 ERROR_CURRENT_DIRECTORY
Attempting to remove current directory.

17 ERROR_NOT_SAME_DEVICE
Not same device.

18 ERROR_NO_MORE_FILES
No more files.

19 ERROR_WRITE_PROTECT
Attempt to write on write-protected diskette.

20 ERROR_BAD_UNIT
Unknown unit.

21 ERROR_NOT_READY
Drive not ready.

22 ERROR_BAD_COMMAND
Unknown command.

23 ERROR_CRC
Data error (CRC).

24 ERROR_BAD_LENGTH
Bad request structure length.

25 ERROR_SEEK
Seek error.

26 ERROR_NOT_DOS_DISK
Unknown media type.

27 ERROR_SECTOR_NOT_FOUND
Sector not found.

28 ERROR_OUT_OF_PAPER
Printer out of paper.

29 ERROR_WRITE_FAULT
Write fault.

30 ERROR_READ_FAULT
Read fault.

31 ERROR_GEN_FAILURE
General failure.

32 ERROR_SHARING_VIOLATION
Sharing violation.

33 ERROR_LOCK_VIOLATION
Lock violation.

34 ERROR_WRONG_DISK
Invalid disk change.

35 ERROR_FCB_UNAVAILABLE
FCB unavailable.

36 ERROR_SHARING_BUFFER_EXCEEDED
Sharing buffer overflow.

37–49 Reserved.

50 ERROR_NOT_SUPPORTED
Network request not supported.

65 Access denied.

73–79 Reserved.

80 ERROR_FILE_EXISTS
File exists.

81 ERROR_DUP_FCB
Reserved.

82 ERROR_CANNOT_MAKE
Cannot make directory entry.

83 ERROR_FAIL_I24
Fail on INT 24.

84 ERROR_OUT_OF_STRUCTURES
Too many redirections.

85 ERROR_ALREADY_ASSIGNED
Duplicate redirection.

86 ERROR_INVALID_PASSWORD
Invalid password.

87 ERROR_INVALID_PARAMETER
Invalid parameter.

88 ERROR_NET_WRITE_FAULT
Network device fault.

89 ERROR_NO_PROC_SLOTS
No process slots available.

90 ERROR_NOT_FROZEN
System error.

91 ERR_TSTOVFL
Timer service table overflow.

92 ERR_TSTDUP
Timer service table duplicate.

93 ERROR_NO_ITEMS
No items to work on.

95 ERROR_INTERRUPT
Interrupted system call.

99 ERROR_DEVICE_IN_USE
Device in use.

100 ERROR_TOO_MANY_SEMAPHORES
User/system open semaphore limit exceeded.

101 ERROR_EXCL_SEM_ALREADY_OWNED
Exclusive semaphore already owned.

102 ERROR_SEM_IS_SET
DosCloseSem found semaphore set.

103 ERROR_TOO_MANY_SEM_REQUESTS
Too many exclusive semaphore requests.

104 ERROR_INVALID_AT_INTERRUPT_TIME
Operation invalid at interrupt time.

105 ERROR_SEM_OWNER_DIED
Previous semaphore owner terminated without freeing semaphore.

106 ERROR_SEM_USER_LIMIT
Semaphore limit exceeded.

107 ERROR_DISK_CHANGE
Insert drive B disk into drive A.

108 ERROR_DRIVE_LOCKED
Drive locked by another process.

109 ERROR_BROKEN_PIPE
Write on pipe with no reader.

110 ERROR_OPEN_FAILED
Open/create failed due to explicit fail command.

111 ERROR_BUFFER_OVERFLOW
Buffer passed to system call too small to hold return data.

112 ERROR_DISK_FULL
Not enough space on the disk.

113 ERROR_NO_MORE_SEARCH_HANDLES
Cannot allocate another search structure and handle.

114 ERROR_INVALID_TARGET_HANDLE
Target handle in DosDupHandle invalid.

115 ERROR_PROTECTION_VIOLATION
Bad user virtual address.

116 ERROR_VIOKBD_REQUEST
Error on display write or keyboard read.

117 ERROR_INVALID_CATEGORY
Category for DevIOCtl not defined.

118 ERROR_INVALID_VERIFY_SWITCH
Invalid value passed for verify flag.

119 ERROR_BAD_DRIVER_LEVEL
Level four driver not found.

120 ERROR_CALL_NOT_IMPLEMENTED
Invalid function called.

121 ERROR_SEM_TIMEOUT
Time out occurred from semaphore API function.

122 ERROR_INSUFFICIENT_BUFFER
Data buffer too small.

123 ERROR_INVALID_NAME
Illegal character or bad file-system name.

124 ERROR_INVALID_LEVEL
Non-implemented level for information retrieval or setting.

125 ERROR_NO_VOLUME_LABEL
No volume label found with DosQFsInfo command.

126 ERROR_MOD_NOT_FOUND
Module handle not found with getprocaddr, getmodhandle.

127 ERROR_PROC_NOT_FOUND
Procedure address not found with getprocaddr.

128 ERROR_WAIT_NO_CHILDREN
DosCwait finds no children.

129 ERROR_CHILD_NOT_COMPLETE
DosCwait children not terminated.

130 ERROR_DIRECT_ACCESS_HANDLE
Handle operation invalid for direct disk-access handles.

131 ERROR_NEGATIVE_SEEK
Attempting seek to negative offset.

132 ERROR_SEEK_ON_DEVICE
Application trying to seek on device or pipe.

133 ERROR_IS_JOIN_TARGET
Drive has previously joined drives.

134 ERROR_IS_JOINED
Drive is already joined.

135 ERROR_IS_SUBSTED
Drive is already substituted.

136 ERROR_NOT_JOINED
Cannot delete drive that is not joined.

137 ERROR_NOT_SUBSTED
Cannot delete drive that is not substituted.

138 ERROR_JOIN_TO_JOIN
Cannot join to a joined drive.

139 ERROR_SUBST_TO_SUBST
Cannot substitute to a substituted drive.

140 ERROR_JOIN_TO_SUBST
Cannot join to a substituted drive.

141 ERROR_SUBST_TO_JOIN
Cannot substitute to a joined drive.

142 ERROR_BUSY_DRIVE
Specified drive is busy.

143 ERROR_SAME_DRIVE
Cannot join or substitute a drive to a directory on the same drive.

144 ERROR_DIR_NOT_ROOT
Directory must be a subdirectory of the root.

145 ERROR_DIR_NOT_EMPTY
Directory must be empty to use join command.

146 ERROR_IS_SUBST_PATH
Path specified is being used in a substitute.

147 ERROR_IS_JOIN_PATH
Path specified is being used in join.

148 ERROR_PATH_BUSY
Path specified is being used by another process.

149 ERROR_IS_SUBST_TARGET
Cannot join or substitute drive having directory that is target of a previous substitute.

150 ERROR_SYSTEM_TRACE
System trace error.

151 ERROR_INVALID_EVENT_COUNT
DosMuxSemWait errors.

152 ERROR_TOO_MANY_MUXWAITERS
System limit of 100 entries reached.

153 ERROR_INVALID_LIST_FORMAT
Invalid list format.

154 ERROR_LABEL_TOO_LONG
Volume label too big.

155 ERROR_TOO_MANY_TCBS
Cannot create another TCB.

156 ERROR_SIGNAL_REFUSED
Signal refused.

157 ERROR_DISCARDED
Segment is discarded.

158 ERROR_NOT_LOCKED
Segment not locked.

159 ERROR_BAD_THREADID_ADDR
Bad thread-identity address.

160 ERROR_BAD_ARGUMENTS
Bad environment pointer.

161 ERROR_BAD_PATHNAME
Bad path name passed to exec.

162 ERROR_SIGNAL_PENDING
Signal already pending.

163 ERROR_UNCERTAIN_MEDIA
ERROR_I24 mapping.

164 ERROR_MAX_THRDS_REACHED
No more process slots.

165 ERROR_MONITORS_NOT_SUPPORTED
ERROR_I24 mapping.

166 ERROR_UNC_DRIVER_NOT_INSTALLED
Default redir return code

167 ERROR_LOCK_FAILED
Locking failed.

168 ERROR_SWAPIO_FAILED
Swap IO failed.

169 ERROR_SWAPIN_FAILED
Swap in failed.

170 ERROR_BUSY
Busy.

180 ERROR_INVALID_SEGMENT_NUMBER
Invalid segment number.

181 ERROR_INVALID_CALLGATE
Invalid call gate.

182 ERROR_INVALID_ORDINAL
Invalid ordinal.

183 ERROR_ALREADY_EXISTS
Shared segment already exists.

184 ERROR_NO_CHILD_PROCESS
No child process to wait for.

185 ERROR_CHILD_ALIVE_NOWAIT
NoWait specified and child alive.

186 ERROR_INVALID_FLAG_NUMBER
Invalid flag number.

187 ERROR_SEM_NOT_FOUND
Semaphore does not exist.

188 ERROR_INVALID_STARTING_CODESEG
Invalid starting code segment, incorrect END (label) directive.

189 ERROR_INVALID_STACKSEG
Invalid stack segment.

190 ERROR_INVALID_MODULETYPE
Invalid module type; dynamic-link library file cannot be used as an application. Application cannot be used as a dynamic-link library.

191 ERROR_INVALID_EXE_SIGNATURE
Invalid EXE signature - file is DOS mode program or improper program.

192 ERROR_EXE_MARKED_INVALID
EXE marked invalid - link detected errors when application created.

193 ERROR_BAD_EXE_FORMAT
Bad EXE format - file is DOS mode program or improper program.

194 ERROR_ITERATED_DATA_EXCEEDS_64K
Iterated data exceeds 64K; more than 64K of data in one of the segments of the file.

195 ERROR_INVALID_MINALLOCSIZE
Invalid minimum allocation size; size is specified to be less than the size of the segment data in the file.

196 ERROR_DYNLINK_FROM_INVALID_RING
Dynamic link from invalid privilege level; privilege level 2 routine cannot link to dynamic-link libraries.

197 ERROR_IOPL_NOT_ENABLED
IOPL not enabled - IOPL set to "NO" in CONFIG.SYS.

198 ERROR_INVALID_SEGDPL
Invalid segment descriptor privilege level - can only have privilege levels of 2 and 3.

199 ERROR_AUTODATASEG_EXCEEDS_64K
Automatic data segment exceeds 64K.

200 ERROR_RING2SEG_MUST_BE_MOVABLE
Privilege level 2 segment must be movable.

201 ERROR_RELOC_CHAIN_XEEDS_SEGLIM
Relocation chain exceeds segment limit.

202 ERROR_INFLOOP_IN_RELOC_CHAIN
Infinite loop in relocation chain segment.

203 ERROR_ENVVAR_NOT_FOUND
Environment variable not found.

204 ERROR_NOT_CURRENT_CTRY
Not current country.

205 ERROR_NO_SIGNAL_SENT
No signal sent - no process in the command subtree has a signal handler.

206 ERROR_FILENAME_EXCED_RANGE
Filename or extension greater than "8.3" characters.

207 ERROR_RING2_STACK_IN_USE
Privilege level 2 stack in use.

208 ERROR_META_EXPANSION_TOO_LONG
Meta (global) expansion is too long.

209 ERROR_INVALID_SIGNAL_NUMBER
Invalid signal number.

210 ERROR_THREAD_1_INACTIVE
Inactive thread.

211 ERROR_INFO_NOT_AVAIL
File system information not available for this file.

212 ERROR_LOCKED
Locked error.

213 ERROR_BAD_DYNALINK
Attempted to execute non-family API in DOS mode.

214 ERROR_TOO_MANY_MODULES
Too many modules.

215 ERROR_NESTING_NOT_ALLOWED
Nesting not allowed.

217 ERROR_ZOMBIE_PROCESS
Zombie process.

218 ERROR_STACK_IN_HIGH_MEMORY
Stack in high memory.

219 ERROR_INVALID_EXITROUTINE_RING
Invalid exit routine ring.

220 ERROR_GETBUF_FAILED
Get buffer failed.

221 ERROR_FLUSHBUF_FAILED
Flush buffer failed.

222 ERROR_TRANSFER_TOO_LONG
Transfer is too long.

228 ERROR_NO_CHILDREN
No child process.

229 ERROR_INVALID_SCREEN_GROUP
Invalid session.

230 ERROR_BAD_PIPE
Nonexistent pipe or bad operation.

231 ERROR_PIPE_BUSY
Pipe is busy.

232 ERROR_NO_DATA
No data available on non-blocking read.

233 ERROR_PIPE_NOT_CONNECTED
Pipe was disconnected by server.

234 ERROR_MORE_DATA
More data is available.

240 ERROR_VC_DISCONNECTED
Session was dropped due to errors.

250 ERROR_CIRCULARITY_REQUESTED
Renaming a directory that would cause a circularity problem.

251 ERROR_DIRECTORY_IN_CDS
Renaming a directory that is in use.

252 ERROR_INVALID_FSD_NAME
Trying to access nonexistent FSD.

253 ERROR_INVALID_PATH
Bad pseudo device.

254 ERROR_INVALID_EA_NAME
Bad character in name, or bad cbName.

255 ERROR_EA_LIST_INCONSISTENT
List does not match its size, or bad EAs in list.

256 ERROR_EA_LIST_TOO_LONG
FEAList > 64K–1 bytes.

257 ERROR_NO_META_MATCH
String doesn't match expression.

259 ERROR_NO_MORE_ITEMS
DosQFSAttach ordinal query.

260 ERROR_SEARCH_STRUC_REUSED
DOS mode findfirst/next search structure reused.

261 ERROR_CHAR_NOT_FOUND
Character not found.

262 ERROR_TOO_MUCH_STACK
Stack request exceeds system limit.

263 ERROR_INVALID_ATTR
Invalid attribute.

264 ERROR_INVALID_STARTING_RING
Invalid starting ring.

265 ERROR_INVALID_DLL_INIT_RING
Invalid DLL INIT ring.

266 ERROR_CANNOT_COPY
Cannot copy.

267 ERROR_DIRECTORY
Used by DOSCOPY in doscall1.

268 ERROR_OPLOCKED_FILE
Oplocked file.

269 ERROR_OPLOCK_THREAD_EXISTS
Oplock thread exists.

270 ERROR_VOLUME_CHANGED
Volume changed.

271–273 Reserved.

274 ERROR_ALREADY_SHUTDOWN
System already shutdown.

275 ERROR_EAS_DIDNT_FIT
EAS didnt fit.

303 ERROR_INVALID_PROCID
Invalid process identity.

304 ERROR_INVALID_PDELTA
Invalid priority delta.

305 ERROR_NOT_DESCENDANT
Not descendant.

306 ERROR_NOT_SESSION_MANAGER
Requestor not session manager.

307 ERROR_INVALID_PCLASS
Invalid P class.

308 ERROR_INVALID_SCOPE
Invalid scope.

309 ERROR_INVALID_THREADID
Invalid thread identity.

310 ERROR_DOSSUB_SHRINK
Cannot shrink segment—DosSubSet.

311 ERROR_DOSSUB_NOMEM
No memory to satisfy request—DosSubAlloc.

312 ERROR_DOSSUB_OVERLAP
Overlap of specified block with an allocated memory—DosSubFree.

313 ERROR_DOSSUB_BADSIZE
Bad size parameter - DosSubAlloc or DosSubFree.

314 ERROR_DOSSUB_BADFLAG
Bad flag parameter—DosSubSet.

315 ERROR_DOSSUB_BADSELECTOR
Invalid segment selector.

316 ERROR_MR_MSG_TOO_LONG
Message too long for buffer.

317 ERROR_MR_MID_NOT_FOUND
Message identity number not found.

318 ERROR_MR_UN_ACC_MSGF
Unable to access message file.

319 ERROR_MR_INV_MSGF_FORMAT
Invalid message file format.

320 ERROR_MR_INV_IVCOUNT
Invalid insertion variable count.

321 ERROR_MR_UN_PERFORM
Unable to perform function.

322 ERROR_TS_WAKEUP
Unable to wake up.

323 ERROR_TS_SEMHANDLE
Invalid system semaphore.

324 ERROR_TS_NOTIMER
No timers available.

326 ERROR_TS_HANDLE
Invalid timer handle.

327 ERROR_TS_DATETIME
Date or time invalid.

328 ERROR_SYS_INTERNAL
Internal system error.

329 ERROR_QUE_CURRENT_NAME
Current queue name does not exist.

330 ERROR_QUE_PROC_NOT_OWNED
Current process does not own queue.

331 ERROR_QUE_PROC_OWNED
Current process owns queue.

332 ERROR_QUE_DUPLICATE
Duplicate queue name.

333 ERROR_QUE_ELEMENT_NOT_EXIST
Queue element does not exist.

334 ERROR_QUE_NO_MEMORY
Inadequate queue memory.

335 ERROR_QUE_INVALID_NAME
Invalid queue name.

336 ERROR_QUE_INVALID_PRIORITY
Invalid queue priority parameter.

337 ERROR_QUE_INVALID_HANDLE
Invalid queue handle.

338 ERROR_QUE_LINK_NOT_FOUND
Queue link not found.

339 ERROR_QUE_MEMORY_ERROR
Queue memory error.

340 ERROR_QUE_PREV_AT_END
Previous queue element was at end of queue.

341 ERROR_QUE_PROC_NO_ACCESS
Process does not have access to queues.

342 ERROR_QUE_EMPTY
Queue is empty.

343 ERROR_QUE_NAME_NOT_EXIST
Queue name does not exist.

344 ERROR_QUE_NOT_INITIALIZED
Queues not initialized.

345 ERROR_QUE_UNABLE_TO_ACCESS
Unable to access queues.

346 ERROR_QUE_UNABLE_TO_ADD
Unable to add new queue.

347 ERROR_QUE_UNABLE_TO_INIT
Unable to initialize queues.

349 ERROR_VIO_INVALID_MASK
Invalid function replaced.

350 ERROR_VIO_PTR
Invalid pointer to parameter.

351 ERROR_VIO_APTR
Invalid pointer to attribute.

352 ERROR_VIO_RPTR
Invalid pointer to row.

353 ERROR_VIO_CPTR
Invalid pointer to column.

354 ERROR_VIO_LPTR
Invalid pointer to length.

355 ERROR_VIO_MODE
Unsupported screen mode.

356 ERROR_VIO_WIDTH
Invalid cursor width value.

357 ERROR_VIO_ATTR
Invalid cursor attribute value.

358 ERROR_VIO_ROW
Invalid row value.

359 ERROR_VIO_COL
Invalid column value.

360 ERROR_VIO_TOPROW
Invalid TopRow value.

361 ERROR_VIO_BOTROW
Invalid BotRow value.

362 ERROR_VIO_RIGHTCOL
Invalid right column value.

363 ERROR_VIO_LEFTCOL
Invalid left column value.

364 ERROR_SCS_CALL
Call issued by other than sm

365 ERROR_SCS_VALUE
Value is not for save or restore.

366 ERROR_VIO_WAIT_FLAG
Invalid wait flag setting.

367 ERROR_VIO_UNLOCK
Screen not previously locked.

368 ERROR_SGS_NOT_SESSION_MGR
Caller not session manager.

369 ERROR_SMG_INVALID_SGID
Invalid session identity.

369 ERROR_SMG_INVALID_SESSION_ID
Invalid session ID.

370 ERROR_SMG_NOSG
No sessions available.

370 ERROR_SMG_NO_SESSIONS
No sessions available.

371 ERROR_SMG_GRP_NOT_FOUND
Session not found.

371 ERROR_SMG_SESSION_NOT_FOUND
Session not found.

372 ERROR_SMG_SET_TITLE
Title sent by shell or parent cannot be changed.

373 ERROR_KBD_PARAMETER
Invalid parameter to keyboard.

374 ERROR_KBD_NO_DEVICE
No device.

375 ERROR_KBD_INVALID_IOWAIT
Invalid I/O wait specified.

376 ERROR_KBD_INVALID_LENGTH
Invalid length for keyboard.

377 ERROR_KBD_INVALID_ECHO_MASK
Invalid echo mode mask.

378 ERROR_KBD_INVALID_INPUT_MASK
Invalid input mode mask.

379 ERROR_MON_INVALID_PARMS
Invalid parameters to DosMon.

380 ERROR_MON_INVALID_DEVNAME
Invalid device name string.

381 ERROR_MON_INVALID_HANDLE
Invalid device handle.

382 ERROR_MON_BUFFER_TOO_SMALL
Buffer too small.

383 ERROR_MON_BUFFER_EMPTY
Buffer is empty.

384 ERROR_MON_DATA_TOO_LARGE
Data record too large.

385 ERROR_MOUSE_NO_DEVICE
Mouse device closed (invalid device handle).

386 ERROR_MOUSE_INV_HANDLE
Mouse device closed (invalid device handle).

387 ERROR_MOUSE_INV_PARMS
Parameters invalid for display mode.

388 ERROR_MOUSE_CANT_RESET
Function assigned and cannot be reset.

389 ERROR_MOUSE_DISPLAY_PARMS
Parameters invalid for display mode.

390 ERROR_MOUSE_INV_MODULE
Module not valid.

391 ERROR_MOUSE_INV_ENTRY_PT
Entry point not valid.

392 ERROR_MOUSE_INV_MASK
Function mask invalid.

393 NO_ERROR_MOUSE_NO_DATA
No valid data.

394 NO_ERROR_MOUSE_PTR_DRAWN
Pointer drawn.

395 ERROR_INVALID_FREQUENCY
Invalid frequency for beep.

396 ERROR_NLS_NO_COUNTRY_FILE
Cannot find COUNTRY.SYS file.

397 ERROR_NLS_OPEN_FAILED
Cannot open COUNTRY.SYS file.

398 ERROR_NLS_NO_CTRY_CODE
Country code not found.

398 ERROR_NO_COUNTRY_OR_CODEPAGE
Country code not found.

399 ERROR_NLS_TABLE_TRUNCATED
Table returned information truncated, buffer too small.

400 ERROR_NLS_BAD_TYPE
Selected type does not exist.

401 ERROR_NLS_TYPE_NOT_FOUND
Selected type not in file.

402 ERROR_VIO_SMG_ONLY
Valid from session manager only.

403 ERROR_VIO_INVALID_ASCIIZ
Invalid ASCIIZ length.

404 ERROR_VIO_DEREGISTER
VioDeRegister not allowed.

405 ERROR_VIO_NO_POPUP
Pop-up window not allocated.

406 ERROR_VIO_EXISTING_POPUP
Pop-up window on screen (NoWait).

407 ERROR_KBD_SMG_ONLY
Valid from session manager only.

408 ERROR_KBD_INVALID_ASCIIZ
Invalid ASCIIZ length.

409 ERROR_KBD_INVALID_MASK
Invalid replacement mask.

410 ERROR_KBD_REGISTER
KbdRegister not allowed.

411 ERROR_KBD_DEREGISTER
KbdDeRegister not allowed.

412 ERROR_MOUSE_SMG_ONLY
Valid from session manager only.

413 ERROR_MOUSE_INVALID_ASCIIZ
Invalid ASCIIZ length.

414 ERROR_MOUSE_INVALID_MASK
Invalid replacement mask.

415 ERROR_MOUSE_REGISTER
Mouse register not allowed.

416 ERROR_MOUSE_DEREGISTER
Mouse deregister not allowed.

417 ERROR_SMG_BAD_ACTION
Invalid action specified.

418 ERROR_SMG_INVALID_CALL
INIT called more than once or invalid session identity.

419 ERROR_SCS_SG_NOTFOUND
New session number.

420 ERROR_SCS_NOT_SHELL
Caller is not shell.

421 ERROR_VIO_INVALID_PARMS
Invalid parameters passed.

422 ERROR_VIO_FUNCTION_OWNED
Save/restore already owned.

423 ERROR_VIO_RETURN
Non-destruct return (undo).

424 ERROR_SCS_INVALID_FUNCTION
Caller invalid function.

425 ERROR_SCS_NOT_SESSION_MGR
Caller not session manager.

426 ERROR_VIO_REGISTER
Vio register not allowed.

427 ERROR_VIO_NO_MODE_THREAD
No mode restore thread in SG.

428 ERROR_VIO_NO_SAVE_RESTORE_THD
No save/rest thread in SG.

429 ERROR_VIO_IN_BG
Function invalid in background.

430 ERROR_VIO_ILLEGAL_DURING_POPUP
Function not allowed during pop-up window.

431 ERROR_SMG_NOT_BASESHELL
Caller is not the base shell.

432 ERROR_SMG_BAD_STATUSREQ
Invalid status requested.

433 ERROR_QUE_INVALID_WAIT
NoWait parameter out of bounds.

434 ERROR_VIO_LOCK
Error returned from Scroll Lock.

435 ERROR_MOUSE_INVALID_IOWAIT
Invalid parameters for IOWait.

436 ERROR_VIO_INVALID_HANDLE
Invalid VIO handle.

437 ERROR_VIO_ILLEGAL_DURING_LOCK
Function not allowed during screen lock.

438 ERROR_VIO_INVALID_LENGTH
Invalid VIO length.

439 ERROR_KBD_INVALID_HANDLE
Invalid KBD handle.

440 ERROR_KBD_NO_MORE_HANDLE
Ran out of handles.

441 ERROR_KBD_CANNOT_CREATE_KCB
Unable to create kcb.

442 ERROR_KBD_CODEPAGE_LOAD_INCOMPL
Unsuccessful code-page load.

443 ERROR_KBD_INVALID_CODEPAGE_ID
Invalid code-page identity.

444 ERROR_KBD_NO_CODEPAGE_SUPPORT
No code page support.

445 ERROR_KBD_FOCUS_REQUIRED
Keyboard focus required.

446 ERROR_KBD_FOCUS_ALREADY_ACTIVE
Calling thread has an outstanding focus.

447 ERROR_KBD_KEYBOARD_BUSY
Keyboard busy.

448 ERROR_KBD_INVALID_CODEPAGE
Invalid code page.

449 ERROR_KBD_UNABLE_TO_FOCUS
Focus attempt failed.

450 ERROR_SMG_SESSION_NON_SELECT
Session is not selectable.

451 ERROR_SMG_SESSION_NOT_FOREGRND
Parent/child session not foreground.

452 ERROR_SMG_SESSION_NOT_PARENT
Not parent of requested child.

453 ERROR_SMG_INVALID_START_MODE
Invalid session start mode.

454 ERROR_SMG_INVALID_RELATED_OPT
Invalid session start related option.

455 ERROR_SMG_INVALID_BOND_OPTION
Invalid session bond option.

456 ERROR_SMG_INVALID_SELECT_OPT
Invalid session select option.

457 ERROR_SMG_START_IN_BACKGROUND
Session started in background.

458 ERROR_SMG_INVALID_STOP_OPTION
Invalid session stop option.

459 ERROR_SMG_BAD_RESERVE
Reserved parameters not zero.

460 ERROR_SMG_PROCESS_NOT_PARENT
Session parent process already exists.

461 ERROR_SMG_INVALID_DATA_LENGTH
Invalid data length.

462 ERROR_SMG_NOT_BOUND
Parent not bound.

463 ERROR_SMG_RETRY_SUB_ALLOC
Retry request block allocation.

464 ERROR_KBD_DETACHED
This call not allowed for detached PID.

465 ERROR_VIO_DETACHED
This call disallowed for detached pid.

466 ERROR_MOU_DETACHED
This call disallowed for detached pid.

467 ERROR_VIO_FONT
No font available to support mode.

468 ERROR_VIO_USER_FONT
User font active.

469 ERROR_VIO_BAD_CP
Invalid code page specified.

470 ERROR_VIO_NO_CP
System displays do not support code page.

471 ERROR_VIO_NA_CP
Current display does not support code page.

472 ERROR_INVALID_CODE_PAGE
Invalid code page.

473 ERROR_CPLIST_TOO_SMALL
Code page list is too small.

474 ERROR_CP_NOT_MOVED
Code page not moved.

475 ERROR_MODE_SWITCH_INIT
Mode switch initialization error.

476 ERROR_CODE_PAGE_NOT_FOUND
Code page not found.

477 ERROR_UNEXPECTED_SLOT_RETURNED
Internal error.

478 ERROR_SMG_INVALID_TRACE_OPTION
Invalid start session trace indicator.

479 ERROR_VIO_INTERNAL_RESOURCE
VIO internal resource error.

480 ERROR_VIO_SHELL_INIT
VIO shell initialization error.

481 ERROR_SMG_NO_HARD_ERRORS
No session manager hard errors.

482 ERROR_CP_SWITCH_INCOMPLETE
DosSetCp unable to set KBD or VIO code page.

483 ERROR_VIO_TRANSPARENT_POPUP
Error during VIO pop-up window.

484 ERROR_CRITSEC_OVERFLOW
Critical section overflow.

485 ERROR_CRITSEC_UNDERFLOW
Critical section underflow.

486 ERROR_VIO_BAD_RESERVE
Reserved parameter is not zero.

487 ERROR_INVALID_ADDRESS
Bad physical address.

488 ERROR_ZERO_SELECTORS_REQUESTED
At least one selector must be requested.

489 ERROR_NOT_ENOUGH_SELECTORS_AVA
Not enough GDT selectors to satisfy request.

490 ERROR_INVALID_SELECTOR
Not a GDT selector.

491 ERROR_SMG_INVALID_PROGRAM_TYPE
Invalid program type.

492 ERROR_SMG_INVALID_PGM_CONTROL
Invalid program control.

493 ERROR_SMG_INVALID_INHERIT_OPT
Bad inherit option.

494 ERROR_VIO_EXTENDED_SG

495 ERROR_VIO_NOT_PRES_MGR_SG

496 ERROR_VIO_SHIELD_OWNED

497 ERROR_VIO_NO_MORE_HANDLES

498 ERROR_VIO_SEE_ERROR_LOG

499 ERROR_VIO_ASSOCIATED_DC

500 ERROR_KBD_NO_CONSOLE

501 ERROR_MOUSE_NO_CONSOLE

502 ERROR_MOUSE_INVALID_HANDLE

503 ERROR_SMG_INVALID_DEBUG_PARMS

504 ERROR_KBD_EXTENDED_SG

505 ERROR_MOU_EXTENDED_SG

506 ERROR_SMG_INVALID_ICON_FILE

# Index

## V

## X

Covers OS/2 2.0-2.1

**Get th… OS/2 documentation code for all the keyboard, mouse, and video functions that have been driving you crazy!**

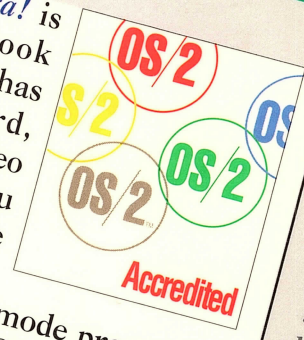
uments the
MOU/VIO
whatever
ot include
for the
s in their
Library.
trend is
al User
here is
creen,
to run
node
VIO
t of
ter

## IBM approved!

## An indispensable addition to your OS/2 programming library

If you waste endless hours poring through online documentation to code your OS/2 applications, this could be the most valuable book purchase you'll make for a long time to come!

*OS/2® Extra!* is the only book available that has the keyboard, mouse, and video functions you need to create and port full-screen character-mode programs that will run in OS/2. Editors Len Dorfman and Marc Neuberger also include a helpful appendix listing the error codes returned by the KBD/MOU/VIO calls and their descriptions.




b
d
op
lik
a
bet
prog
the ha
Thi
based
docum
provided
We ha
to orient

OPERATING SYSTEMS
Beginner/Intermediate

$19.95

0893

ISBN 0-8306-4567-5


Windcrest®/McGraw-Hill
Blue Ridge Summit, PA 17294-0850